

# Java Web

## 轻量级开发全体验

—— 邓子云 / 著 ——

电子工业出版社  
Publishing House of Electronics Industry  
北京·BEIJING

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间：可以联系群主获取更多大型企业内部技术教程。

## 内 容 简 介

本书共 2 篇（分为 18 章）：基础篇、框架技术篇。全书内容遵循“循序渐进”的原则，逐步深入，理论联系实际，内容通俗易懂，涵盖了当前 Java Web 开发所流行的众多开发技术，注重项目实战，致力于培养技术娴熟、能上手开发软件系统的 Java Web 程序员。

随书的光盘带有书中所有实例和实战项目的源代码，以供读者参考学习。

本书的适用面较广，初、中、高级读者均可阅读，可作为中职、高职、本科计算机专业或相近专业的 Java Web 开发或 JSP 课程的教材，也可作为 Java Web 培训班的教材，亦可供对 JSP 感兴趣的读者自学使用。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。  
版权所有，侵权必究。

## 图书在版编目（CIP）数据

Java Web 轻量级开发全体验 / 邓子云著. —北京：电子工业出版社，2012.1  
ISBN 978-7-121-14749-4

I. ①J… II. ①邓… III. ①JAVA 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字（2011）第 201950 号

策划编辑：孙学瑛

责任编辑：葛 娜

印 刷：北京市顺义兴华印刷厂

装 订：三河市双峰印刷装订有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×1092 1/16 印张：34.75 字数：860 千字

印 次：2012 年 1 月第 1 次印刷

印 数：4000 册 定价：69.80 元（含光盘 1 张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 [zltz@phei.com.cn](mailto:zltz@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线：（010）88258888。



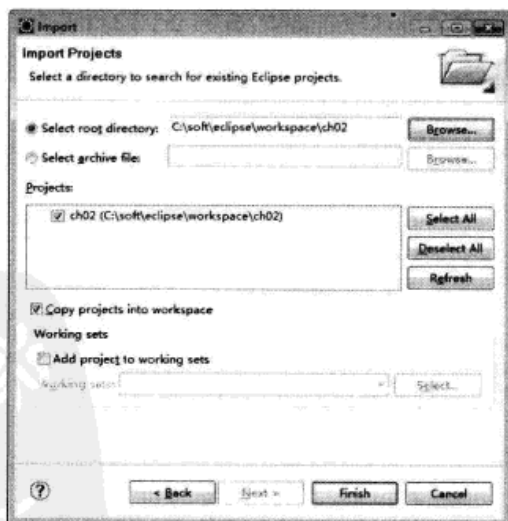
# 光盘内容与使用说明

## 光盘内容

为方便读者阅读本书和调试程序，随书附带的光盘中包含全书所有实例的源代码和实战项目的源代码。第 2 章至第 18 章中的源代码根据章节存放在光盘中，以章名命名为一个 Web 应用，比如第 2 章的 Web 应用为“ch02”，使用 Eclipse 的“Import”功能导入已存在的工程即可使用。数据库的库文件存放在光盘的“数据库”目录中，在 SQL Server 组中附加即可。书中需要用到的软件组件包在各章的 Web 应用中已经自带了，不必再进行下载。

## 导入 Web 应用的方法

选择 Eclipse 的“File”→“Import...”菜单，进入“Import”对话框，在树形菜单中选择“General”→“Existing Projects into Workspace”，然后单击“Next”按钮，如图附 1-1 所示。



图附 1-1 导入 Web 应用

# 目 录

## 基础篇

<b>第 1 章 JSP 技术概述</b>	<b>2</b>	2.3 开发工具的使用	16
1.1 程序网络计算模式	2	2.3.1 搭建 Web 系统框架	17
1.1.1 C/S 模式	2	2.3.2 开发一个 JSP 页面	19
1.1.2 B/S 模式	3	2.4 小结	22
1.1.3 两种模式的比较分析	3	2.5 练习	23
1.2 B/S 模式技术介绍	4	<b>第 3 章 Web 开发基础</b>	<b>24</b>
1.2.1 CGI	4	3.1 HTML	24
1.2.2 ASP 与 ASP.NET	5	3.1.1 什么是 HTML	24
1.2.3 PHP	7	3.1.2 URL	24
1.2.4 JSP	7	3.1.3 HTML 结构	25
1.2.5 JSP 与其他 B/S 模式 技术的比较	7	3.1.4 HTML 标记	25
1.3 小结	8	3.1.5 表单	26
1.4 练习	8	3.2 JavaScript	27
<b>第 2 章 安装与配置环境</b>	<b>9</b>	3.2.1 何谓 JavaScript	27
2.1 应用服务器介绍	9	3.2.2 加入 JavaScript	27
2.1.1 Tomcat	9	3.2.3 JavaScript 对象	28
2.1.2 WebLogic	10	3.3 Web 信息交互	29
2.1.3 IBM WebSphere	11	3.3.1 表单信息交互	29
2.2 JSP 运行环境的安装与配置	11	3.3.2 用正则表达式验证 提交的数据	33
2.2.1 JDK 的安装与配置	11	3.4 小结	37
2.2.2 Tomcat 7 的安装与配置	13	3.5 练习	37
2.2.3 Eclipse 的安装与配置	14	<b>第 4 章 JSP 语法</b>	<b>38</b>
		4.1 JSP 的基本结构	38

4.2	数据类型	39
4.2.1	数据类型概述	40
4.2.2	标识符	40
4.2.3	简单数据类型	41
4.2.4	数组	45
4.2.5	类	46
4.2.6	String 类	51
4.2.7	StringBuffer 类	57
4.3	运算符与表达式	61
4.3.1	算术运算与表达式	61
4.3.2	关系运算与表达式	63
4.3.3	布尔运算与表达式	63
4.3.4	位运算与表达式	64
4.4	程序控制逻辑	64
4.4.1	选择分支	64
4.4.2	循环	67
4.5	Java 程序片	69
4.6	程序注释	70
4.7	JSP 指令	71
4.7.1	page 指令	71
4.7.2	include 指令	72
4.8	JSP 动作指令	73
4.8.1	include 动作指令	74
4.8.2	forward 动作指令	75
4.8.3	param 动作指令	76
4.8.4	useBean 动作指令	77
4.8.5	setProperty 动作指令	78
4.8.6	getProperty 动作指令	79
4.9	JSP 中的中文字符处理	79
4.10	小结	81
4.11	练习	81

## 第 5 章 JSP 的内置对象 82

5.1	内置对象概述	82
5.2	request 对象	83
5.2.1	request 对象的方法	83
5.2.2	获得表单数据	88
5.3	response 对象	91

5.3.1	response 对象的方法	91
5.3.2	使用 Cookie	93
5.3.3	response 对象重定向	94
5.3.4	定时刷新页面	95
5.4	session 对象	97
5.4.1	session 对象的方法	97
5.4.2	猜字母游戏	100
5.5	application 对象	102
5.5.1	application 对象的方法	102
5.5.2	计数器	103
5.6	out 对象	105
5.6.1	out 对象的方法	105
5.6.2	用 out 对象输出表格	106
5.7	小结	107
5.8	练习	107

## 第 6 章 JSP 中数据库的使用 108

6.1	SQL 基础	108
6.1.1	表操作	109
6.1.2	查询语句	110
6.1.3	插入、更新与删除语句	112
6.1.4	存储过程	113
6.2	JDBC	114
6.2.1	JDBC 工作原理	114
6.2.2	JDBC 的四种驱动	114
6.2.3	ODBC 数据源	115
6.2.4	SQL Server 的 JDBC 安装	117
6.2.5	JDBC 接口	118
6.3	查询记录	135
6.3.1	顺序查询	135
6.3.2	移动查询	137
6.3.3	参数查询	139
6.3.4	模糊查询	141
6.3.5	综合查询	143
6.4	追加记录	145
6.5	删除记录	149
6.6	更新记录	152
6.7	在 ResultSet 中修改数据	158



6.7.1	追加记录	158
6.7.2	删除记录	159
6.7.3	更新记录	160
6.8	分页显示记录	161
6.9	调用存储过程	166
6.10	事务处理	167
6.11	连接其他数据库	169
6.11.1	Oracle	169
6.11.2	MySQL	169
6.11.3	Informix	169
6.11.4	Sybase	169
6.11.5	AS400	170
6.12	连接池技术	170
6.12.1	什么是 Connection Pool	170
6.12.2	Tomcat 7 上 Connection Pool 的配置	170
6.12.2	Connection Pool 应用实例	172
6.13	得到元数据	173
6.13.1	DatabaseMetaData	173
6.13.2	ResultSetMetaData	174
6.13.3	得到表名和列名	174
6.14	小结	176
6.15	练习	176

## 第 7 章 JSP 中 JavaBean 的应用 178

7.1	什么是 JavaBean	178
7.2	编写 JavaBean	179
7.3	编译和部署 JavaBean	180
7.3.1	编译 JavaBean	180
7.3.2	部署 JavaBean	182
7.4	小结	186
7.5	练习	187

## 第 8 章 Servlet 技术 188

8.1	什么是 Servlet	188
8.2	Servlet 的工作原理	189
8.2.1	Servlet 的生命周期	189

8.2.2	Servlet 相关的类及方法	189
8.2.3	部署 Servlet	191
8.3	用 Servlet 获取表单数据	193
8.4	小结	194
8.5	练习	194

## 第 9 章 JSTL 应用开发 195

9.1	JSTL 技术概述	195
9.1.1	JSTL 介绍	195
9.1.2	安装 JSTL	196
9.1.3	标签书写语法约定	196
9.1.4	标签的分类	196
9.2	核心标签	197
9.2.1	表达式相关的核心标签	197
9.2.2	流程控制核心标签	200
9.2.3	迭代核心标签	201
9.2.4	URL 相关的核心标签	205
9.3	国际化处理标签	207
9.3.1	国际化类处理标签	208
9.3.2	消息类处理标签	209
9.3.3	数字日期格式化类处理标签	210
9.4	SQL 标签	218
9.4.1	设置数据源	218
9.4.2	查询数据	219
9.4.3	显示查询结果	221
9.4.4	更新数据	224
9.4.5	日期型数据处理	225
9.4.6	事务处理	226
9.5	函数标签	228
9.5.1	求长度函数	229
9.5.2	大小写转换函数	229
9.5.3	求子串函数	229
9.5.4	去空白函数	230
9.5.5	替换函数	230
9.5.6	查找函数	230
9.5.7	拆分与组合函数	231
9.5.8	XML 符号转换函数	232

9.6 小结 .....	233
9.7 练习 .....	233

## 第 10 章 EL 表达式 234

10.1 EL 简介 .....	234
10.1.1 运算符 .....	234
10.1.2 保留字 .....	237
10.1.3 变量查找范围 .....	237
10.1.4 自动类型转换 .....	237
10.2 EL 隐含对象 .....	239
10.3 用 EL 操作 JavaBean .....	242
10.4 小结 .....	244
10.5 练习 .....	244

## 第 11 章 常用开发功能实现 245

11.1 操作 XML 文件 .....	245
11.1.1 XML 概述 .....	245
11.1.2 XML 文件的结构 .....	246
11.1.3 DTD 文档 .....	247
11.1.4 XML Schema .....	252
11.1.5 JDOM .....	261
11.2 上传和下载文件 .....	267
11.2.1 jspSmartUpload 常用 的 API .....	268
11.2.2 上传文件 .....	273
11.2.3 下载文件 .....	280
11.3 制作 Web 报表与图形 .....	281
11.3.1 JavaReport 简介 .....	281
11.3.2 JavaReport 常用的 API .....	282
11.3.3 如何开发 Web 图形与 报表 .....	289
11.3.4 开发统计图 .....	293
11.3.5 开发 Web 报表 .....	296
11.4 生成验证码 .....	299
11.5 发送和接收邮件 .....	303
11.5.1 下载与安装 JavaMail .....	303

11.5.2 JavaMail 常用的 API .....	303
11.5.3 发送邮件 .....	309
11.5.4 接收邮件 .....	313

11.6 小结 .....	317
11.7 练习 .....	317

## 第 12 章 基于 JSP 实现报到 管理系统 318

12.1 系统需求 .....	318
12.1.1 系统业务需求 .....	318
12.1.2 系统功能需求 .....	319
12.2 系统设计 .....	320
12.2.1 系统设计思想 .....	320
12.2.2 数据库系统设计思路 .....	321
12.2.3 数据库系统的实现 .....	322
12.3 系统开发框架搭建 .....	328
12.3.1 在 Eclipse 中搭建 Web 应用的开发框架 .....	328
12.3.2 设计总体的页面效果 .....	329
12.4 系统各功能点的实现 .....	334
12.4.1 用户登录功能的实现 .....	334
12.4.2 专业基础数据管理 功能的实现 .....	338
12.4.3 录取学生名册基础数据 管理功能的实现 .....	342
12.4.4 其他基础数据管理 功能的实现 .....	350
12.4.5 学生报到状况查询 功能的实现 .....	351
12.4.6 用户管理功能的实现 .....	351
12.4.7 报到分班功能的实现 .....	352
12.4.8 收费情况登记功能的 实现 .....	356
12.4.9 宿舍分配功能的实现 .....	360
12.5 小结 .....	361



## 框架技术篇

### 第 13 章 Struts 2 框架技术 364

- 13.1 Struts 2 框架介绍 ..... 364
  - 13.1.1 MVC 模式 ..... 364
  - 13.1.2 Struts 2 原理 ..... 365
  - 13.1.3 安装与配置 Struts ..... 367
- 13.2 应用 Struts 2 ..... 368
  - 13.2.1 用 Struts 2 实现用户登录功能 ..... 368
  - 13.2.2 解决中文乱码的问题 ..... 374
- 13.3 国际化应用程序 ..... 375
  - 13.3.1 为用户登录功能加入国际化处理 ..... 375
  - 13.3.2 查找资源文件的顺序 ..... 377
  - 13.3.3 参数化字符串 ..... 578
- 13.4 OGNL 表达式 ..... 378
  - 13.4.1 Struts 2 对表达式的支持 ..... 378
  - 13.4.2 使用 OGNL 表达式 ..... 379
  - 13.4.3 值堆栈 ..... 383
  - 13.4.4 OGNL 与集合 ..... 384
- 13.5 Struts 2 标签 ..... 385
  - 13.5.1 标签属性值的设置 ..... 386
  - 13.5.2 控制标签 ..... 387
  - 13.5.3 数据标签 ..... 391
  - 13.5.4 表单标签 ..... 396
  - 13.5.5 非表单用户界面标签 ..... 406
- 13.6 数据校验 ..... 406
  - 13.6.1 服务端和客户端数据校验 ..... 407
  - 13.6.2 字段校验 ..... 410
  - 13.6.3 复杂的数据类型转换 ..... 416
- 13.7 小结 ..... 420

### 第 14 章 基于 Struts 2 实现报到管理系统 421

- 14.1 系统设计思想 ..... 421

- 14.2 系统开发框架搭建 ..... 422
  - 14.2.1 在 Eclipse 中搭建 Web 应用的开发框架 ..... 422
  - 14.2.2 准备相关的配置文件与包 ..... 424
- 14.3 系统各功能点的实现 ..... 425
  - 14.3.1 用户登录功能的实现 ..... 426
  - 14.3.2 专业基础数据管理功能的实现 ..... 430
  - 14.3.3 录取学生名册基础数据管理功能的实现 ..... 433
  - 14.3.4 其他基础数据管理功能的实现 ..... 442
  - 14.3.5 学生报到状况查询功能的实现 ..... 442
  - 14.3.6 报到分班功能的实现 ..... 445
  - 14.3.7 收费情况登记功能的实现 ..... 448
  - 14.3.8 宿舍分配功能的实现 ..... 452
- 14.4 小结 ..... 454

### 第 15 章 Hibernate 4 持久化技术 455

- 15.1 Hibernate 介绍 ..... 455
  - 15.1.1 Hibernate 的作用 ..... 455
  - 15.1.2 Hibernate Core for Java ..... 456
  - 15.1.3 Hibernate 的核心 API ..... 456
- 15.2 安装与配置 Hibernate 4 ..... 457
  - 15.2.1 下载 Hibernate 4 ..... 457
  - 15.2.2 配置 Hibernate 4 ..... 457
- 15.3 一个简单的 Hibernate Web 应用 ..... 459
- 15.4 持久化对象 ..... 464
  - 15.4.1 Session 接口 ..... 464
  - 15.4.2 映射配置 ..... 466
- 15.5 关联关系映射 ..... 474
  - 15.5.1 单向多对一关联 ..... 474

15.5.2	双向多对一关联 .....	477
15.5.3	一对一关联 .....	480
15.5.4	多对多关联 .....	481
15.6	HQL 语言 .....	481
15.6.1	select update delete .....	482
15.6.2	where 子句 .....	482
15.6.3	order by 子句 .....	483
15.6.4	group by 子句 .....	483
15.7	Struts 2 与 Hibernate 4 的集成 .....	483
15.7.1	集成的策略 .....	483
15.7.2	集成的实现 .....	484
15.8	小结 .....	488
15.9	练习 .....	489

## 第 16 章 基于 Struts 2+Hibernate 4 实现报到管理系统 490

16.1	系统设计思想 .....	490
16.2	系统开发框架搭建 .....	491
16.2.1	在 Eclipse 中搭建 Web 应用的开发框架 .....	491
16.2.2	准备 Hibernate 相关的配置文件与包 .....	492
16.2.3	设计 POJO 类与映射配置文件 .....	494
16.2.4	开发会话工厂类 .....	499
16.3	系统各功能点的实现 .....	500
16.3.1	用户登录功能的实现 .....	500
16.3.2	专业基础数据管理功能的实现 .....	501
16.3.3	录取学生名册基础数据管理功能的实现 .....	503
16.3.4	其他基础数据管理功能的实现 .....	506
16.3.5	学生报到状况查询功能的实现 .....	506

16.3.6	报到分班功能的实现 .....	507
16.3.7	收费情况登记功能的实现 .....	509
16.3.8	宿舍分配功能的实现 .....	511
16.4	小结 .....	513
16.5	练习 .....	513

## 第 17 章 Spring 3 框架技术 514

17.1	Spring 介绍 .....	514
17.1.1	Spring 的框架结构 .....	514
17.1.2	理解 IoC 与 DI .....	515
17.2	控制反转技术 .....	516
17.2.1	容器的基本原理 .....	516
17.2.2	XML 配置文件格式 .....	517
17.2.3	实例化容器 .....	517
17.2.4	下载并开发一个简单的 Spring 应用 .....	518
17.2.5	XML 配置文件解析 .....	521
17.2.6	使用容器 .....	523
17.3	集成 Struts 2、Hibernate 4 与 Spring 3 .....	524
17.3.1	集成前的环境准备 .....	524
17.3.2	集成示例与剖析 .....	525
17.4	小结 .....	531
17.5	练习 .....	531

## 第 18 章 基于 SSH 实现报到管理系统 532

18.1	系统设计思想 .....	532
18.1.1	改进思路 .....	532
18.1.2	系统配置文件 .....	533
18.2	系统实现 .....	536
18.3	小结 .....	537
18.4	练习 .....	537

# 实例目录

## 基础篇

### 第2章 安装与配置环境 9

【实例 2-1】新建第一个 JSP 页面 .....19

### 第3章 Web 开发基础 24

【实例 3-1】HTML 与 JavaScript  
交互实例 .....30

【实例 3-2】正则表达式验证  
数据实例 .....35

### 第4章 JSP 语法 38

【实例 4-1】JSP 程序的基本结构 .....38

【实例 4-2】简单数据类型综合  
应用实例 .....42

【实例 4-3】包装类综合应用实例 .....44

【实例 4-4】数组应用实例 .....46

【实例 4-5】字符截取程序实例 .....53

【实例 4-6】查找字符串程序实例 .....55

【实例 4-7】StringBuffer 综合  
应用程序实例 .....59

【实例 4-8】省略显示长  
字符串实例 .....60

【实例 4-9】算术表达式综合  
运用实例 .....62

【实例 4-10】switch 语句示例 ..... 66

【实例 4-11】循环应用综合实例 ..... 68

【实例 4-12】一个简单的计数器 ..... 69

【实例 4-13】include 指令应用  
实例 ..... 73

【实例 4-14】forward 应用程序示例 .. 75

【实例 4-15】param 应用程序示例 .... 76

【实例 4-16】中文字符处理  
程序示例 ..... 80

### 第5章 JSP 的内置对象 82

【实例 5-1】request 常用方法的  
应用 ..... 86

【实例 5-2】获得表单数据 ..... 88

【实例 5-3】页面重定向程序示例 ..... 95

【实例 5-4】定时刷新页面程序  
示例 ..... 95

【实例 5-5】记住会话的用户名 ..... 98

【实例 5-6】猜字母游戏 ..... 100

【实例 5-7】网站计数器 ..... 104

【实例 5-8】用 out 对象输出表格 .... 106

### 第6章 JSP 中数据库的使用 108

【实例 6-1】顺序查询数据库表  
中的数据 ..... 135



【实例 6-2】移动查询 .....	137
【实例 6-3】参数查询 .....	139
【实例 6-4】模糊查询 .....	141
【实例 6-5】综合查询 .....	143
【实例 6-6】追加记录 .....	145
【实例 6-7】删除记录 .....	149
【实例 6-8】更新记录 .....	152
【实例 6-9】分页显示记录 .....	161
【实例 6-10】调用存储过程 .....	166
【实例 6-11】事务处理 .....	168
【实例 6-12】利用连接池访问 数据库 .....	172
【实例 6-13】得到表名和列名 .....	174

## 第 7 章 JSP 中 JavaBean 的应用 178

【实例 7-1】JSP 中的 JavaBean 应用 .....	183
【实例 7-2】用 HTML 表单设置 JavaBean 的属性值 .....	185

## 第 8 章 Servlet 技术 188

【实例 8-1】一个简单的 Servlet .....	192
【实例 8-2】用 Servlet 获取表单 数据 .....	193

## 第 9 章 JSTL 应用开发 195

【实例 9-1】表达式相关的核心 标签综合实例 .....	199
【实例 9-2】迭代核心标签综合 实例 .....	203
【实例 9-3】使用 URL 标签 .....	207

【实例 9-4】使用数字与日期 格式处理标签 .....	217
【实例 9-5】运用 SQL 标签 查询数据 .....	221
【实例 9-6】运用 SQL 标签做事务 处理 .....	226
【实例 9-7】函数标签应用示例 .....	232

## 第 10 章 EL 表达式 234

【实例 10-1】使用 EL 表达式 .....	238
【实例 10-2】使用 EL 表达式获得 表单中的数据 .....	241
【实例 10-3】通过 EL 表达式使用 JavaBean .....	242

## 第 11 章 常用开发功能实现 245

【实例 11-1】user.xml 文件的 DTD 文档 .....	250
【实例 11-2】使用 JDOM 创建 user.xml .....	264
【实例 11-3】用 JDOM 方式读取 XML 文件 .....	266
【实例 11-4】上传文件 .....	273
【实例 11-5】上传文件到数据库 .....	276
【实例 11-6】下载文件 .....	280
【实例 11-7】开发 Web 统计图 .....	293
【实例 11-8】开发 Web 统计报表 .....	296
【实例 11-9】生成彩色验证码 .....	299
【实例 11-10】发送邮件 .....	309
【实例 11-11】接收邮件 .....	313

## 框架技术篇

## 第 13 章 Struts 2 框架技术 364

【实例 13-1】用 Struts 2 实现用户登录 功能 .....	368
---	-----

【实例 13-2】为用户登录功能加入 国际化处理 .....	375
【实例 13-3】OGNL 表达式使用 示例 .....	380

【实例 13-4】<s:action>标签使用 示例 .....	392
【实例 13-5】用 XML 配置文件做简单的 服务端和客户端数据 校验 .....	407
【实例 13-6】使用 Struts 2 中的 字段校验 .....	411
【实例 13-7】将字符串转换为 对象 .....	416

## 第 15 章 Hibernate 4 持久化技术 455

【实例 15-1】hibernate.cfg.xml 的 配置文件示例 .....	458
【实例 15-2】一个简单的 Hibernate Web 应用 .....	459
【实例 15-3】映射配置示例 .....	470

【实例 15-4】多对一关联程序 示例 .....	474
【实例 15-5】双向多对一关联程序 示例 .....	477
【实例 15-6】用 Struts 2+Hibernate 3 集成实现用户登录 功能 .....	484

## 第 17 章 Spring 3 框架技术 514

【实例 17-1】一个简单的 IoC 应用 示例 .....	518
【实例 17-2】用 Struts 2+Hibernate 4+Spring 3 集成改进 用户登录功能 .....	525



# 基 础 篇

- ◆ 第 1 章 JSP 技术概述
- ◆ 第 2 章 安装与配置环境
- ◆ 第 3 章 Web 开发基础
- ◆ 第 4 章 JSP 语法
- ◆ 第 5 章 JSP 的内置对象
- ◆ 第 6 章 JSP 中数据库的使用
- ◆ 第 7 章 JSP 中 JavaBean 的应用
- ◆ 第 8 章 Servlet 技术
- ◆ 第 9 章 JSTL 应用开发
- ◆ 第 10 章 EL 表达式
- ◆ 第 11 章 常用开发功能实现
- ◆ 第 12 章 基于 JSP 实现报到管理系统

# 01

## JSP 技术概述

---



JSP 已成为当今最为流行的网络编程语言之一，广泛地运用于电子商务、电子政务及各行业的软件中。JSP 是一种动态网页技术，具有跨平台性、运行效率高、上手容易等优点。只要具备程序设计的基本知识，学习 JSP 将会变得容易，所以许多程序员纷纷学习或转向学习 JSP 程序设计。

本章从网络程序的计算模式谈起，讲解现在网络软件开发中最常用的两种计算模式——C/S 模式、B/S 模式，并对其做了对比分析；接着介绍 B/S 模式常见的几种技术，包括 CGI、ASP 与 ASP.NET、PHP、JSP，并将 JSP 与其他技术进行比较。

这一章旨在带领新入门或是刚转向编写 JSP 的程序员了解 JSP 的基础知识，是全书的一个引子。学习完本章后，应当对网络编程技术的两种模式及四种动态网页技术有所了解。

### 1.1 程序网络计算模式

随着网络技术的不断发展，单机的软件程序已难以满足人们网络计算的需求，各种各样的网络计算模式应运而生。其中 C/S 与 B/S 模式是网络计算模式中运用得最多的两种计算模式。

#### 1.1.1 C/S 模式

C/S (Client/Server, 客户机/服务器) 方式的网络计算模式，其工作分别由服务器和客户机完成。

服务器负责管理数据库的访问，为多个客户程序管理数据，并对数据库进行检索和排序，此外还要对客户机/服务器网络结构中的数据库安全层加锁，进行保护。

客户机负责与用户的交互，收集用户信息，通过网络向服务器请求对诸如数据库、电子表格或文档等信息的处理工作。

可见，在 C/S 模式中，资源明显不对等，是一种“胖客户机”或“瘦服务器”结构。

最简单的 C/S 模式数据库应用由两部分组成，即客户应用程序和数据库服务器程序。两者



可分别称为前台程序与后台程序。运行数据库服务器程序的机器，称为应用服务器，服务器程序启动后，就随时等待响应客户程序发来的请求；客户程序在客户使用的计算机上运行，客户使用的计算机称之为客户机。当需要对数据库中的数据进行访问时，客户程序就自动寻找服务器程序，并向其发出请求，服务器程序根据预定的规则进行应答，送回结果。应用的形式如图 1-1 所示。

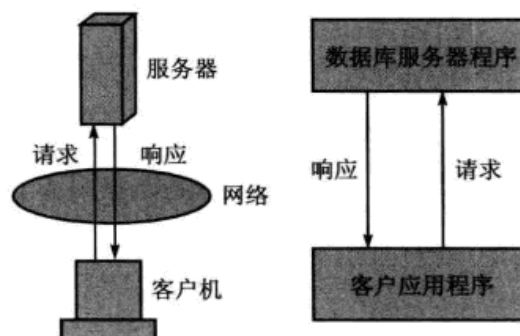


图 1-1 简单的 C/S 模式应用

### 1.1.2 B/S 模式

B/S（Browser/Server，浏览器/服务器）方式的网络结构，在客户端统一采用如 Netscape 和 IE 一类的浏览器，通过 Web 浏览器向 Web 服务器提出请求，由 Web 服务器对数据库进行操作，并将结果传回客户端。

在 B/S 体系结构系统中，用户通过浏览器向分布在网络上的许多服务器发出请求，服务器对浏览器的请求进行处理，将用户所需的信息返回到浏览器。B/S 结构简化了客户机的工作，客户机上只须配置少量的客户端软件即可。但是服务器将担负更多的工作，对数据库的访问和应用程序的执行都将在服务器上完成。即当浏览器发出请求后，其数据请求、加工、返回结果以及动态网页生成等工作全部由 Web 服务器完成。

这种三层体系结构如图 1-2 所示。

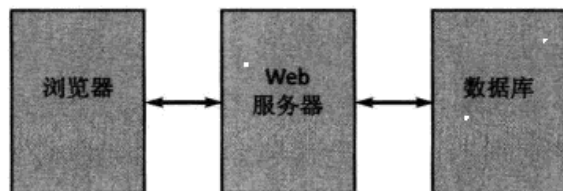


图 1-2 B/S 模式的三层应用

### 1.1.3 两种模式的比较分析

#### 1. 开发和维护成本

C/S 开发和维护成本较高。对不同客户端需要开发不同的程序，且应用程序的安装、修改和升级，均需要在所有的客户机上进行。而 B/S，客户端只需有通用的浏览器，所有的维护与升级工作都是在服务器上执行，无须对客户端进行任何改变，因而大大降低了开发和维护的成本。

#### 2. 客户端负载

C/S 的客户端具有显示与处理数据的功能，负载重。随着应用系统的功能越来越复杂，客户端的应用程序也变得越来越庞大。B/S 的客户端把事务处理逻辑部分给了服务器，客户端只需进行显示即可，俗称为“瘦”客户机。

### 3. 可移植性

C/S 移植困难, 因为不同开发工具开发的应用程序, 一般情况下互不兼容, 难以移植到其他平台上运行。对于 B/S, 在客户端安装的是通用浏览器, 不存在移植性问题。

### 4. 用户界面

C/S 用户的界面是由客户端所安装的软件决定的, 因此用户界面各不相同; 而 B/S 通过通用的浏览器访问应用程序, 其浏览器的界面统一, 使用时类似于浏览网页。

### 5. 安全性

C/S 适用于专人使用的系统, 可以通过严格的管理派发软件, 适用于安全性要求较高的专用应用软件; 而 B/S 适用于交互性要求较多, 使用人数较多, 安全性要求不是很高的应用环境。

综上所述, 这两种开发模式都是网络环境下的开发模式, B/S 相对于 C/S 具有更多的优势, 现如今大量的应用开始转移到应用 B/S 的模式, 许多软件公司争相开发 B/S 版本的软件。由于 Internet 逐步走进人们的日常生活当中, 电子商务进一步应用的需求, 客户简便化的使用要求等对加速推广使用 B/S 模式起到了推波助澜的作用。

## 1.2 B/S 模式技术介绍

B/S 模式下的编程技术有许多种, 这里列出比较常见的几种, 以供对比分析。

### 1.2.1 CGI

CGI (Common Gateway Interface, 通用网关接口) 技术的原理如图 1-3 所示。

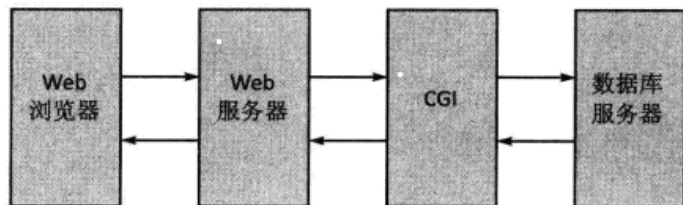


图 1-3 CGI 技术原理框架图

首先, 客户端 (即 Web 浏览器) 根据某资源的 URL (Uniform Resource Locator, 统一资源定位器) 向 Web 服务器提出请求; 然后, Web 服务器的 HTTP Daemon (守护进程) 将此请求的参数通过标准输入 `stdin` 和环境变量传递给指定的 CGI 程序, 并启动此应用程序进行处理, 如要存取数据库服务器上数据库的数据, 则向数据库服务器发出处理请求, 由数据库服务器将执行结果返回给 CGI 程序; 最后, CGI 程序把处理结果通过标准输出 `stdout` 返回给 HTTP Daemon 进程, 再由 HTTP Daemon 进程返回给客户端, 通过浏览器解析执行, 将最终结果在用户面上显示。

CGI 允许 Web 服务器运行外部应用程序, 通过外部程序来访问数据库等一些外部资源, 并产生 HTML 文档给浏览器。但每次请求 CGI 程序都要重新启动程序, 影响了响应的速度, 且 CGI



程序不能被多个客户请求共享,又影响了各种资源的使用效率。为了克服 CGI 的这些缺点,一些 Web 服务器厂商开发出了专用的 API (Applications Program Interface, 应用程序接口),这样就允许程序员编写程序来扩展服务器的功能。API 相对于 CGI 与 Web 服务器的结合更加紧密,占用的资源较少,提高了运行的效率,其安全性与维护性更好。但是开发 API 应用程序比开发 CGI 应用程序复杂得多,要求程序员掌握更多的计算机软件知识,且各种 API 之间的兼容性不好,业界没有一个统一的标准,使得 API 程序只能工作在专用的 Web 服务器与操作系统之上。

编写 CGI 的程序设计语言有许多种,常用的有 C、Perl、Visual C++ 等,由于对程序员的要求较高且编写与调试比其他的编程技术困难,故近年来在基于 B/S 的信息系统工程实践中很少采用。

### 1.2.2 ASP 与 ASP.NET

ASP (Active Server Pages), 是基于微软 Windows 平台的动态页面开发技术,可以用 VBScript 或 JavaScript 语言来编写,支持 COM/DCOM 构件模型,易学易用,开发效率高。

IIS (Internet Information Serve, 因特网信息服务) 用于建立 NT 系统的 Web 服务器,它是在 NT 的各个版本中进行捆绑销售的组件,与 NT 集成完美,提供了 WWW (World Wide Web, 万维网)、FTP (File Transfer Protocol, 文件传输协议)、SMTP (Simple Mail Transfer Protocol, 简单邮件传输协议) 等各种服务。

开发 ASP 程序时,常常需要先设计静态网页的栏目、格式和版式,并形成 HTML 语言,接着在 HTML 的基础上添加脚本程序,形成 ASP 程序后再实现动态的 Web 网页。在 ASP 的程序开发中,它所用的脚本语言是 JavaScript 和 VBScript,在默认情况下使用的是 VBScript,它是 VB 程序设计语言的一个子集,语法与 VB 基本相同。

ASP 中的 ADO 对象用来执行与数据库相关的操作,ADO 以 OLEDB 或 ODBC 的方式访问数据库,在 .NET 版本中这一对象称为 ADO.NET。

ASP.NET 是 ASP 技术的下一代开发技术。借助于 .NET Framework,可以在 ASP.NET Web 开发中使用各种各样的开发语言,如 VB.NET、C#、C++ 等多种高级语言。

.NET 平台中集成了一系列的技术,如: COM+、XML 等,整个 .NET 平台包括四部分产品。

(1) .NET 开发工具。.NET 开发工具由 .NET 语言 (C#、VB.NET)、一个集成的 IDE (Visual Studio.NET)、类库和通用语言运行时间 (CLR) 构成。

(2) .NET 专用服务器。.NET 专用服务器由一些 .NET 企业服务器组成,如 SQL Server、Exchange、BizTalk 等。这些企业服务器可以为数据存储、E-mail、B2B 电子商务等专用服务提供支持。

(3) .NET Web 服务。虽然 Web Service 不是 .NET 所特有的,但 .NET 为 Web Service 提供了强有力的支持。开发者使用 .NET 平台可以很容易地开发 Web Service。

(4) .NET 设备。作为同 J2ME 竞争的部分,.NET 还为手持设备提供了支持,如手机等。

完整的 .NET 平台涵盖了 JVM、J2SE 和 J2EE 全部的内容。.NET 平台出现的时间较短,大多数读者对 .NET 底层的了解少于 Java 虚拟机。

Microsoft .NET 平台包括 5 个部分,如图 1-4 所示。

(1) 操作系统是 .NET 平台的基础,在操作系统方面,Microsoft 有着强大的开发能力,目前的 .NET 平台可以在包括 Windows Vista、Windows 7 等多个 Microsoft 提供的操作系统中运行。



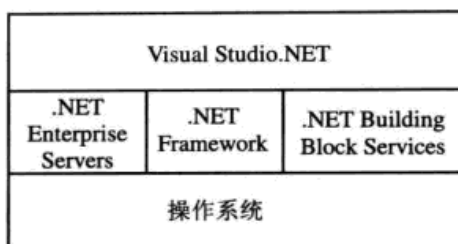


图 1-4 Microsoft .NET 平台

(2) .NET Enterprise Servers 提供了一系列的.NET 服务器产品, 包括: Application Center、BizTalk Server、Commerce Server 等一系列产品。通过这些产品可以缩短构建大型企业应用系统的周期。

(3) .NET Building Block Services 指的是一些成型的服务, 如 Microsoft 提供的 NET Passport 服务。.NET 的开发者可以以付费的方式直接将这些服务集成在自己的应用程序中。

(4) .NET Framework 位于整个.NET 平台的中央, 它不但是技术界讨论的热点, 也是本章比较的重点。.NET Framework 为开发.NET 应用提供了底层的支持, 如 CLR 等。事实上, 即使没有位于顶层的 Visual Studio.NET, 只要有了.NET Framework, 开发者一样可以开发.NET 应用程序。

(5) Visual Studio.NET 是.NET 应用程序的集成开发环境, 它位于.NET 平台的顶端。Visual Studio.NET 是一个强大的开发工具集合, 里面集成了一系列.NET 开发工具, 如 C#.NET、VB.NET、XML Schema Editor 等。

.NET Framework 中引入一系列的新技术和新概念, 图 1-5 给出了.NET Framework 的结构图。其中核心的部分就是通用语言运行时间——CLR。CLR 是.NET 程序的执行引擎, .NET 的众多优点也是由 CLR 所赋予的。CLR 同 JVM 的功能类似, 提供了单一的运行环境。任何.NET 应用程序都会被最终编译为 IL (Intermediate Language, 中间语言), 并在这个统一的环境中运行。也就是说 CLR 可以用于任何针对它的编程语言, 这也就是.NET 的多语言支持。CLR 还负责.NET 应用程序的内存管理、对象生命期的管理、线程管理、安全等一系列的服务。

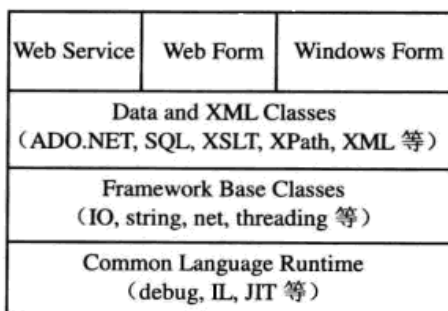


图 1-5 .NET Framework 的组成结构

除了 CLR 外, .NET 提供了.NET 类库, 每一种.NET 语言都可以使用该类库。基本类库中包含了大量的类供开发者使用, 除此之外, 使用某种.NET 语言开发的类可以被其他的.NET 语言直接使用, 从而充分利用各种语言的优点。这也就是说我们可以使用 VB.NET 书写 UI (User Interface, 用户接口) 相关的内容, 而底层的计算功能是用 C++开发。.NET Framework 还对命

名空间提供了支持，熟悉 Java 的程序员一定非常欣赏 Java 中清晰的类的层次结构，.NET Framework 中的命名空间与之类似，非常适合组织大规模的类的层次结果。如 System.Data，或者由开发者自定的 Abc.Accounting.Service。

### 1.2.3 PHP

PHP (Personal Home Pages) 是一种服务器端的脚本语言，嵌入在 HTML 中，它可以在多种平台上运行。

PHP 的语法与 C 语言、Java 语言的语法极为相似，但也有自己独特的语法。它具有庞大的函数库，这主要是因为它是开源式的，源代码完全公开，每个使用 PHP 的程序员都可以加入自己的函数库，从而实现更多的功能，PHP 支持几乎所有的数据库。

但是 PHP 对组件的支持不够，扩展性较差，常与免费的数据库系统 MySQL 一起来构建中小型 B/S 应用的网站或信息系统。

### 1.2.4 JSP

JSP 是 Java Server Pages 的缩写，由 Sun 公司（现被 Oracle 公司收购）倡导，于 1999 年推出，正日益成为开发 Web 动态网站重要而快速有效的开发技术。

JSP 充分利用了 Java 技术的优势，具有极强的扩展能力和良好的伸缩性，与开发平台无关，这源于 Java 的“一次编写，到处运行”的特点，同时也是一项安全的技术。它具有良好的动态页面与静态页面分离的能力，因而正逐渐成为 Internet 开发的主流技术。

JDBC 是 JSP 的数据库连接技术，为其提供了良好的数据库访问能力。

### 1.2.5 JSP 与其他 B/S 模式技术的比较

JSP 相对于其他 B/S 模式下的动态网页技术有诸多的优势，因此它被认为是未来最有发展前途的技术。

#### 1. 跨平台性

ASP 只能在 Windows 平台下运行，而 JSP 基于强大的 Java 语言，几乎可以在所有的操作系统平台上运行，被广泛地运用在 Windows、UNIX、Linux 中。

由于历史的原因，UNIX 的优势依然存在，但越来越多的编程爱好者喜欢使用 Linux，所以生产 Linux 操作系统的商家不断地发行 Linux 的新版本，使其界面更加人性化，功能正日益强大，Internet 上的很多服务都用 Linux 平台。因此，JSP 在这方面占有很大的优势。

JSP 可从一个平台移植到另一个平台，JSP 和 JavaBean 不必重新编译，因为 Java 字节码都是标准的字节码，与平台无关。一些软件公司采取了在 Windows 下开发，Linux 下安装与调试的开发方式，或直接在 Linux 下进行开发，又让 JSP 技术进一步深入人心。

#### 2. 一次编写，到处运行

JSP 拥有 Java 语言“一次编写，到处运行”的特点，所有 JSP 页面都将被 JSP 引擎编译成



Java Servlet，此时就具有了 Java 技术的所有优点，而其他的 B/S 模式技术则没有这个优点。

### 3. 编写容易，快速上手

学习 JSP 将成为一件令程序员感觉很惬意的事情，当然如果有 HTML 及 Java 语言的基础则更好。Web 程序员在网页制作人员设计的 HTML 页面的基础上，找到需要插入 JSP 程序的地方编写 JSP 程序，就可成为 JSP 页面。

### 4. 可重用性

可以将处理逻辑封装在 JavaBeans 或 EJB（Enterprise JavaBeans）组件中，再通过调用 JSP 将处理的结果显示出来。一方面使得开发组件的开发人员可以专注于组件开发；另一方面使编写 JSP 的开发人员可以在多处使用组件，而不必关心其实现细节；而且修改组件只须改动组件内部的设计而不必更改 JSP 代码。这样，大大提高了系统的可重用性，在这个项目中设计的组件在以后的项目中还可以继续使用。

PHP、CGI 技术在可重用性上与 JSP 是无法相提并论的，ASP 和 ASP.NET 支持的组件技术有限，ASP.NET 的构件虽然十分丰富，开发效率较高，但毕竟是基于 Windows 平台的，程序的运行平台受到了限制。而 JSP 不同，构件资料非常丰富，相当多的还是开源的，操作系统平台也没有什么限制。

### 5. 数据库连接技术

Java 程序通过 JDBC（Java Database Connectivity）驱动程序与数据库连接，大部分数据库都带有 JDBC 驱动程序，JDBC-ODBC 的方式提供了 JDBC 与 ODBC 驱动连接的桥梁。大多数的数据库系统带有 ODBC 驱动，这使得 Java 程序可以访问大多数的数据库系统，包括 Oracle、Sybase、Informix、MySQL、Microsoft SQL Server、MS Access 等。

## 1.3 小结

本章介绍了网络程序开发的两种计算模式——C/S 与 B/S 模式。它们各有千秋，用于不同的场合。C/S 适用于专人使用、安全性要求较高的系统；B/S 适用于交互性比较频繁的场合，容易被人们所接受，备受用户和软件开发者的青睐。

B/S 模式下的动态网页技术主要有 CGI、ASP 与 ASP.NET、PHP、JSP 等。其中 JSP 基于 Java 技术，跨平台性好，“一次编写，到处运行”，编写容易，程序员可以快速上手，另外，其可重用性好，连接数据库使用 JDBC 驱动，支持大多数的数据库系统，目前已成为开发 B/S 系统的主流技术。

## 1.4 练习

1. 试比较 C/S 与 B/S 模式的优缺点。
2. 试对比分析各种 B/S 模式下的动态网页技术。

# 02

## 安装与配置环境

编写 JSP 程序第一步就是要搭建环境，本章将引导读者一步一步搭建一个 JSP 运行的基本环境，是初学者要学习的基础知识。若对 JSP 运行环境的安装与配置比较熟悉的读者可跳过此章，直接学习本章之后的内容。

本章首先介绍当前使用 JSP 开发出的应用较广泛的三种 Web 服务器；接着讲述如何安装和配置 JSP 的运行环境；然后在安装与配置好环境后，使用开发工具来编写一个简单的 JSP 页面并运行它。

### 2.1 应用服务器介绍

应用服务器在基于 Internet/Intranet 的应用中起到了举足轻重的作用，它是资源共享、网络通信等分布式应用的基石。国内外的软件公司纷纷推出自己的应用服务器产品，在支持 JSP 的应用服务器中又以 Tomcat、WebLogic、WebSphere 这三种在国内应用较多。

#### 2.1.1 Tomcat

Tomcat 是 Apache-Jakarta 的一个免费、开放源码的子项目，是一个支持 JSP 和 Servlet 技术的容器，它同时又是一个 Web 服务器软件。

Tomcat 很受广大程序员的欢迎，因为它运行时占用的系统资源小，扩展性好，支持负载均衡与邮件服务等开发应用系统常用的功能，而且它还在不断地改进和完善，任何一个感兴趣的程序员都可以更改或在其中加入新的功能。Tomcat 是一个小型的轻量级应用服务器，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试 JSP 程序的首选。

从 <http://tomcat.apache.org> 上可以下载到 Tomcat 的安装程序、源代码及相关的文档，在本书中使用最新的版本 Tomcat 7.0.19（此后简称为 Tomcat 7）。由于 Tomcat 不能单独使用，因此在安装之前必须先行安装 JDK（Java Development Kit）。如图 2-1 所示的网站，在左边的 Download 区中可以下载 Tomcat。单击左边“Download”菜单下的“Tomcat 7.0”超链接，进

入 <http://tomcat.apache.org/download-70.cgi> 页面，选择相应的软件产品（如 Tomcat 中某个版本的软件）即可进行下载。

建议读者下载 Core 下的 zip 版，可根据不同的操作系统来选择。比如 Windows 32 位的操作系统就选择“32-bit Windows zip (pgp, md5)”。

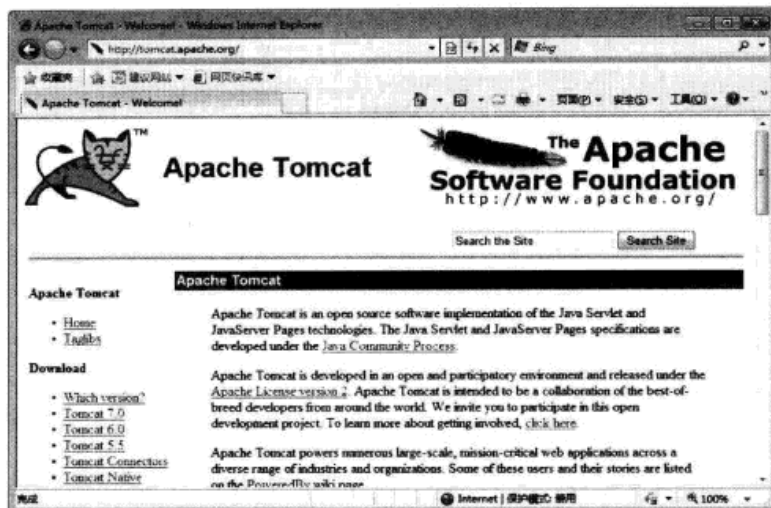


图 2-1 <http://tomcat.apache.org> 网站

## 2.1.2 WebLogic

WebLogic 原是 BEA 公司的产品，目前已转入到 Oracle 公司旗下，现在最新的版本是 11g 版，可从如下的地址得到相关的信息和下载包：<http://www.oracle.com/us/corporate/Acquisitions/bea/index.html>，如图 2-2 所示。

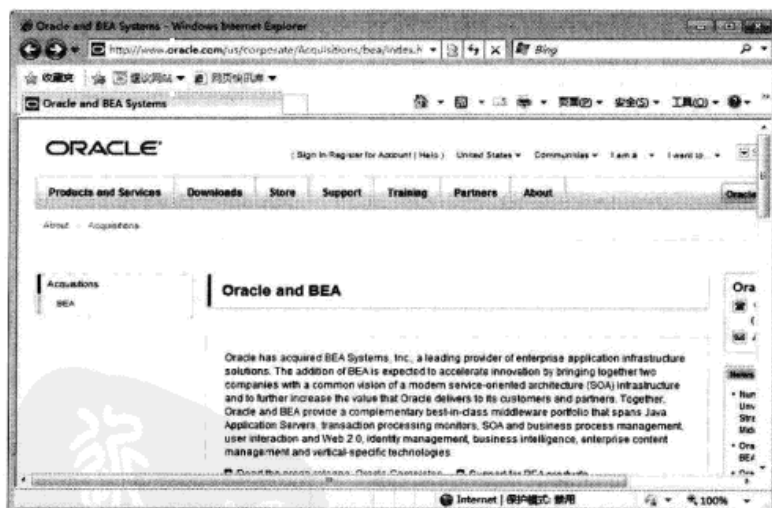


图 2-2 WebLogic 所在的下载网站

WebLogic Server 支持企业级、分布式的 Web 应用，支持包括 JSP、Servlet、EJB 在内的 J2EE 体系中的技术，并提供必要的服务（如事务处理服务），支持集群技术。WebLogic Server 功能特别强大，操作简单、界面友好，在电子商务应用中被大量采用。



### 2.1.3 IBM WebSphere

IBM WebSphere 软件产品系列包括 WebSphere Performance Pack、Cache Manager、Studio、Web 应用开发工具以及 WebSphere Application Server。其中 WebSphere Application Server 是基于 Java™ 的应用环境，用于建立、部署和管理 Internet 和 Intranet Web 应用程序。目前这一整套产品已进行了扩展，以适应 Web 应用程序服务器的需要，范围从简单到高级直到企业级。

WebSphere Application Server 是一款行业领先的 Web 应用服务器，能够为企业应用提供安全、可靠、可扩展的高效运行平台。它支持行业内最广泛的平台，能够为不同类型的应用提供不同的解决方案，消除了对所有应用管理一刀切的方法。它使用基于开放标准的编程模型，包括 Java EE 6、OSGi 应用、Web 2.0 和 Mobile、Java Batch、XML、Service Component Architecture (SCA)、Communications Enabled Applications (CEA)、Session Initiation Protocol (SIP) 和动态脚本。

## 2.2 JSP 运行环境的安装与配置

下面以 JDK 7、Tomcat 7 为例讲述 Web 应用环境的搭建步骤，读者可跟随着一一起做。

### 2.2.1 JDK 的安装与配置

在安装 Tomcat 之前，必须先安装 JDK，可以在 Oracle 公司的网站 <http://www.oracle.com/us/sun/index.htm> 上免费下载，如图 2-3 所示。

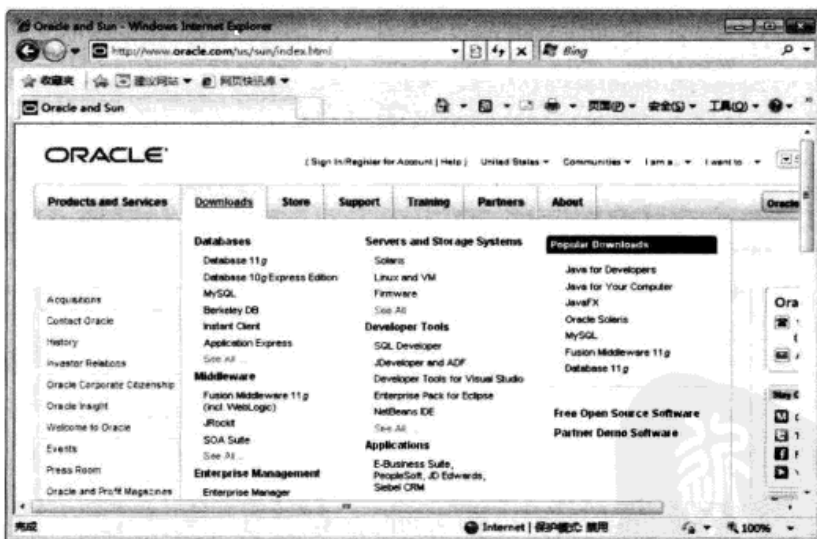


图 2-3 <http://www.oracle.com/us/sun> 网站

将鼠标移动到上边的导航菜单中的“Downloads”上，会出现软件列表，其中在“Popular Downloads”下就有“Java for Developers”，点击即可进入 JDK 的下载界面。进入后可选择所需要的 JDK 版本，这里选择 JDK 7，32 位 Windows 适配的 JDK 7 详细的下载地址为：

<http://download.oracle.com/otn-pub/java/jdk/7/jdk-7-windows-i586.exe>

如果是 64 位 Windows，请使用以下地址：

<http://download.oracle.com/otn-pub/java/jdk/7/jdk-7-windows-x64.exe>

其他操作系统请注意选择对应的版本。

双击下载的 JDK 安装程序 `jdk-7-windows-i586.exe` 或 `jdk-7-windows-x64.exe`，进入安装界面，如图 2-4 所示，单击“下一步(N) >”按钮。接下来要选择安装在硬盘中的目录，单击“更改(A) ...”按钮可更改安装目录，这里更改为“`c:\jdk7`”，具体应用时可根据需要进行设定；单击“下一步(N) >”按钮，进入安装进度对话框；安装完成后会显示如图 2-5 所示的对话框，单击“完成(F)”按钮结束 JDK 7 的安装。

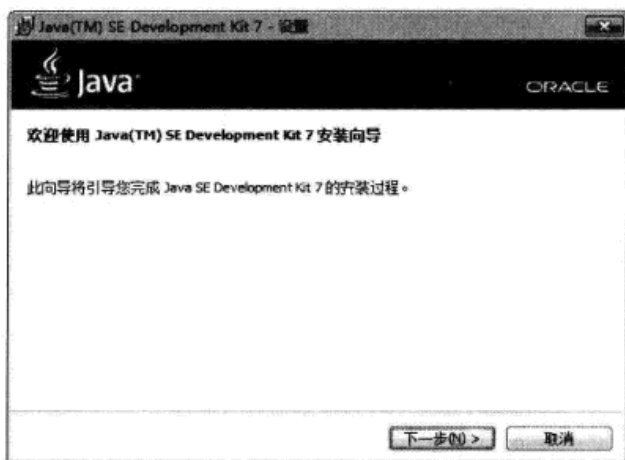


图 2-4 JDK 7 的安装界面

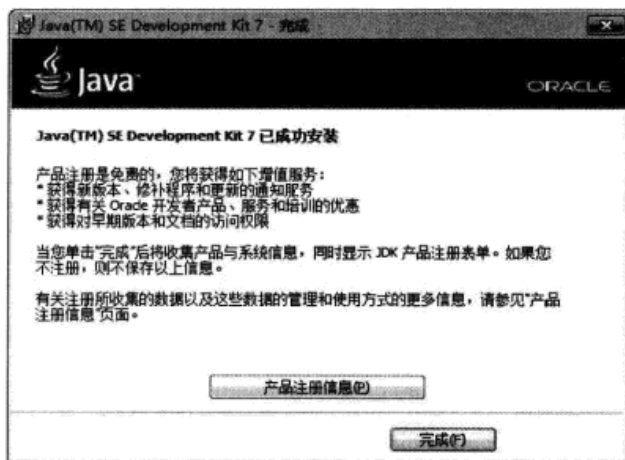


图 2-5 JDK 7 安装完成

安装完 JDK 后，还需要配置两个系统变量。安装完 JDK 后，需要在系统中设置 `PATH` 和 `JAVA_HOME` 这两个环境变量。`PATH` 变量用于指出可执行文件所在的目录，这样在输入命令时就不必再输入冗长的路径了；`JAVA_HOME` 用于指出 JDK 的安装目录，Eclipse、Tomcat 等开发工具都需要用到这个环境变量。

设置环境变量的方法是在桌面上“我的电脑”图标上单击鼠标右键，从快捷菜单中选择“属性(R)”，弹出“系统属性”对话框，选择“高级”选项卡，单击“环境变量(N)”按钮即可进入“环境变量”对话框。

“环境变量”对话框中上面的用户变量是指当前操作系统登录的用户的变量设置，仅对当前用户有效；下面的系统变量则对所有用户有效。为保证通用性，建议读者在“系统变量”中设置环境变量。

如果没有 `PATH` 系统变量就新建一个，如果有就编辑，在后面加入如下的设置：

```
;c:\jdk7\bin
```

“`c:\jdk7`”是指 JDK 的安装目录，前面加个“`;`”表示区分 `PATH` 的各个值。请读者根据实际安装情况修改。“`bin`”目录下存放有 JDK 的可执行文件，如编译 Java 源文件而使用的 `javac` 等。在 `PATH` 中有了路径后就可以直接在命令窗口中输入和使用 JDK 的可执行文件了。

用同样的方法再增加（或修改）`JAVA_HOME` 环境变量，指向“`c:\jdk7`”，请读者根据实际安装情况修改。



## 2.2.2 Tomcat 7 的安装与配置

在 Tomcat 的官方网站得到 Tomcat 7 的压缩文件后,将其用解压缩软件如 WinZip 或 WinRAR 解压至指定的目录,这里解压至 d:\tomcat7,实际应用可根据需要而定。

Tomcat 7 发布的程序版本有 3 种: zip 版、tar.gz 版和 Windows Service Installer 版。在 Windows 操作系统中,Windows Service Installer 版是一个 exe 文件,可根据向导提示安装,此处不再赘述;zip 版无须安装,直接复制解压缩目录下的所有文件至指定目录即可。这里推荐使用 zip 版,因为 Windows Service Installer 版虽然安装简单,但在安装过程中会修改操作系统的注册表,当经过多次安装 Tomcat 后,会出现一些不可预料的错误;而 zip 版无须设置,解压后即可使用。执行 Tomcat 7 安装目录 bin 子目录下的 startup.bat 程序就可启动 Tomcat 7 服务器,启动后如图 2-6 所示。

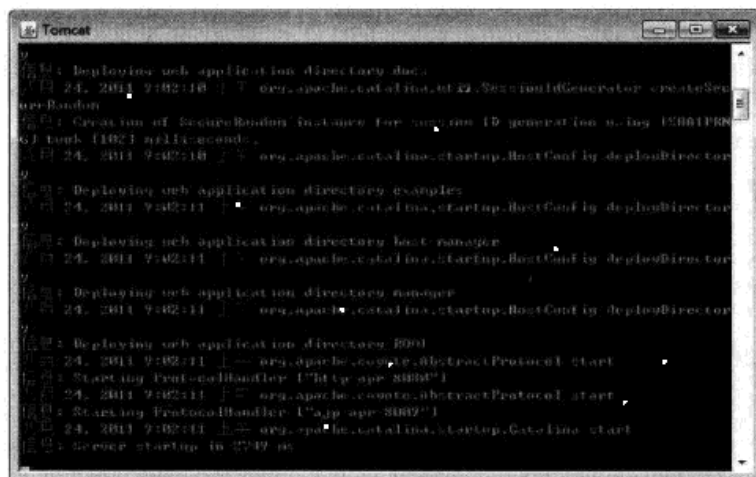


图 2-6 启动 Tomcat 7 服务器

接下来,将测试 Tomcat 7 服务器。打开浏览器,在地址栏中输入 <http://localhost:8080> 或 <http://127.0.0.1:8080>,127.0.0.1 与 localhost 均代表本机。输入地址后出现如图 2-7 所示的效果。

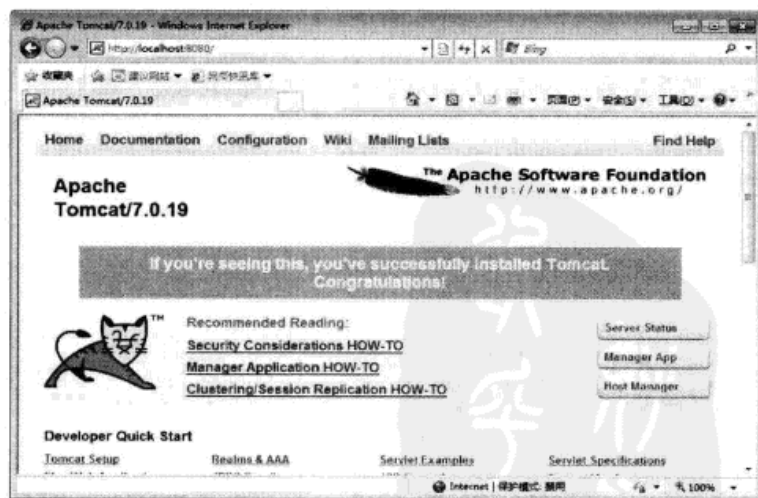


图 2-7 安装 Tomcat 7 后测试是否运行成功



Tomcat 默认的 Web 服务端口号是 8080，IE 浏览器默认的 HTTP 服务端口号是 80，如果将 Tomcat 的 Web 服务端口号改为 80，则在访问服务器时不必再输入端口号了。下面讲解怎么更改 Tomcat 的 Web 服务端口号。

在 Tomcat 的目录中有一个叫做 conf 的子目录，进入后打开其中的 server.xml 文件，可以使用任一文本编辑器，一般使用记事本即可。在 server.xml 中找到如下代码：

```
<Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  redirectPort="8443" />
```

如果觉得查找比较困难，可用记事本“编辑”菜单下的“查找”工具，查找“8080”，以进行定位查找，很快就可以找到上述的文字。

将这段文字中的 port="8080"更改为 port="80"即可。重新启动 Tomcat 服务器，再次测试 Tomcat 服务器时将不需要输入端口号。

### 2.2.3 Eclipse 的安装与配置

Eclipse 是一款免费、开源的集成开发工具，特别适用于 Java 程序的开发。本书的所有实例都将基于 Eclipse 来开发、实现。编写 Java 程序，Eclipse 不是必须的，读者也可以使用任意文本编辑工具，但使用 Eclipse 这样的集成开发工具将大大提高开发的效率。

通过网址“<http://www.eclipse.org>”可进入 Eclipse 的官方网站，如图 2-8 所示。



图 2-8 Eclipse 的官方网站

单击“Download Eclipse”按钮进入下载主页面。其实也可以直接输入“<http://www.eclipse.org/downloads>”进入 Eclipse 的下载主页面。

Eclipse 开发工具有很多种。“Eclipse IDE for Java Developers”专用于 Java 应用程序的开发，带有 Java IDE、CVS 客户端、XML 编辑器等工具，但对 Java Web 应用系统的开发支持不够，需

要另外加装插件。“Eclipse IDE for Java EE Developers”适合进行 Java 企业级应用系统的开发，特别是 Java Web 应用系统的开发，但此工具软件需要 JDK1.5 或更高版本的 JDK 作为支持。

这里下载“Eclipse IDE for Java EE Developers”。

解压缩后，在目录中有一个文件 `eclipse.exe`，双击它，即会弹出如图 2-9 所示的对话框。

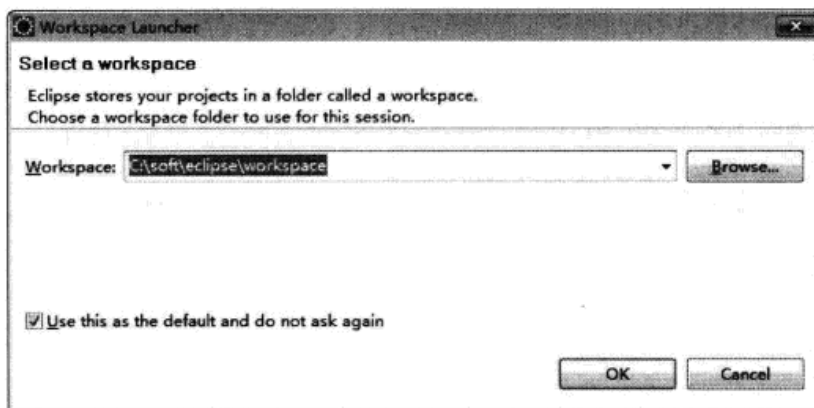


图 2-9 设置 Eclipse 的工作区

这个对话框用于设置 Eclipse 的工作区工作目录，如果选中“Use this as the default and do not ask again”前的复选框，则下次启动 Eclipse 不会再弹出这个对话框，而采用第一次设置的工作区目录。单击“OK”按钮完成设置，进入 Eclipse，如图 2-10 所示。

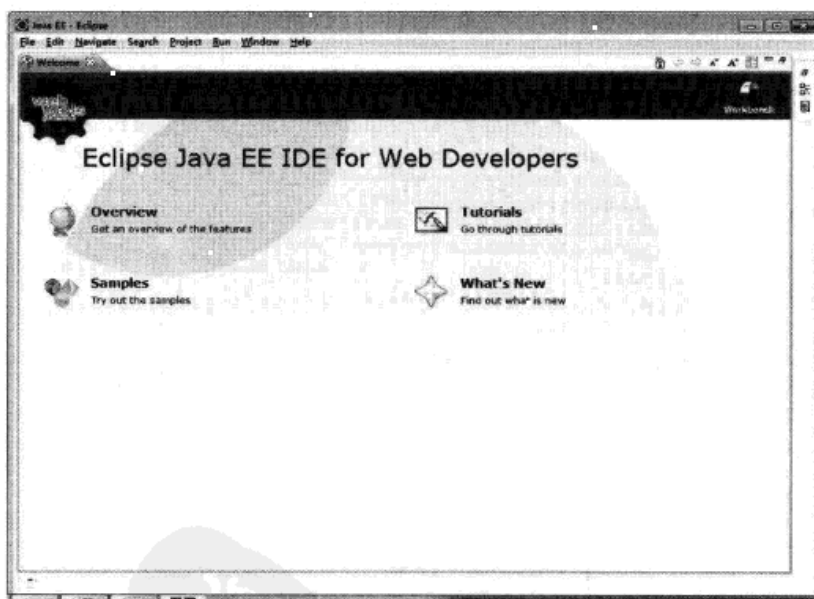


图 2-10 第一次进入 Eclipse

关闭 Welcome 对话框（Welcome 子标题栏右上角有关闭按钮）即可进入到 Eclipse 的工作界面了。

安装了 Eclipse 和 Tomcat 后，这两者还没有关联起来。用 Eclipse 编写的程序都需要有运行环境，而 Eclipse 本身只是一种集成开发工具，并不是程序运行的容器，因此需要在 Eclipse 中对服务器的运行时环境做出一些配置，以方便开发人员在 Eclipse 中直接调试 Web 应用中的程序。



在 Eclipse 中选择“Window”→“Preferences...”，弹出“Preferences”对话框。在“Preferences”对话框左边的树形菜单中选中“Server”→“Runtime Environments”，如图 2-11 所示。

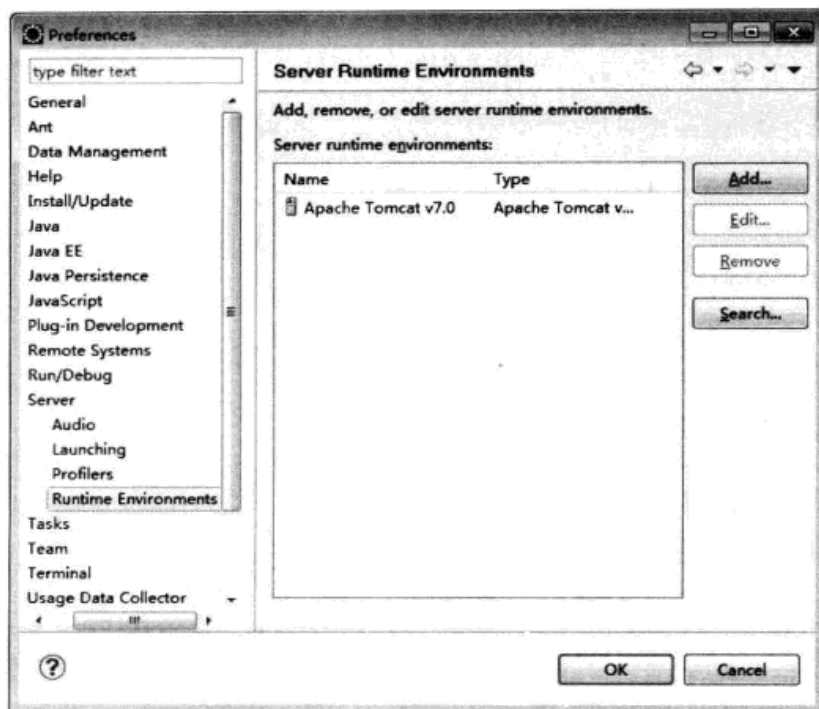


图 2-11 安装服务器运行时环境

从图 2-11 来看，Eclipse 还没有和任何服务器运行环境相关联。单击右边的“Add...”按钮，弹出“New Server Runtime Environment”对话框，选择自己想要的 Web 服务中间件软件，这里选中“Apache Tomcat v7.0”，然后单击“Next”按钮，进入“Tomcat Server”配置步骤。“Name”用于设置名称，读者可自行修改，“Tomcat installation directory”中指出 Tomcat 的安装目录，JRE 中指出系统安装的 JDK，Eclipse 会自动根据系统变量 JAVA\_HOME 找到 JDK，并以 JDK 安装目录名称作为 JRE 的名称。

单击“Finish”按钮完成服务器运行时环境的配置，读者即可以发现在如图 2-11 所示的对话框中会增加一条记录，即为配置成功的服务器运行时的环境。

## 2.3 开发工具的使用

考虑到在学习 JSP 编程时，学生所使用的计算机配置不高，且较少进行重量级程序的开发（如 EJB），可以使用一些轻量的软件工具辅助编程。如 UltraEdit 对程序会有彩色分色显示，对查看源程序有所帮助；编写 JSP 文件可用 Dreamweaver，因为 JSP 中的脚本程序常嵌入在 HTML 代码中，用此工具可以在适当的程序插入点进行较为准确的定位，以减少查找程序的时间，且编写 HTML 和 JSP 代码的功能强大；编写 JavaBean 代码可以使用 Eclipse 软件，它是开源的软件，网上免费提供，且与版本管理软件 CVS 结合较好，在进行 Java 程序的团队式开发时使用较多。当然如果允许，还是建议使用 Eclipse 这样的 IDE 开发工具。



### 2.3.1 搭建 Web 系统框架

开发 Web 系统的第一步就是要搭建起 Web 应用的开发框架，在本章之后的每章都需要搭建一个 Web 应用来开发程序。如果对 Java Web 开发非常熟练，也可以手工搭建开发框架，但是效率较低，而且容易出错。有 Eclipse 就方便多了，它能自动生成开发框架，并自动生成一些通用的配置。

双击“eclipse.exe”文件进入 Eclipse。选择 Eclipse 的“File”→“New”→“Project”菜单，弹出“New Project”对话框；在树形的文件夹菜单中选择“Web”→“Dynamic Web Project”菜单，单击“Next”按钮，进入“New Dynamic Web Project”对话框，如图 2-12 所示，这个对话框用于创建一个新的动态 Web 工程。静态的 Web 工程是不含 JSP 页面的，只是 HTML 网页；动态 Web 工程则是指一个 Java Web 应用，其中可以有 JSP 页面，也可以有 Web 应用自己的类、lib 库、配置文件等。

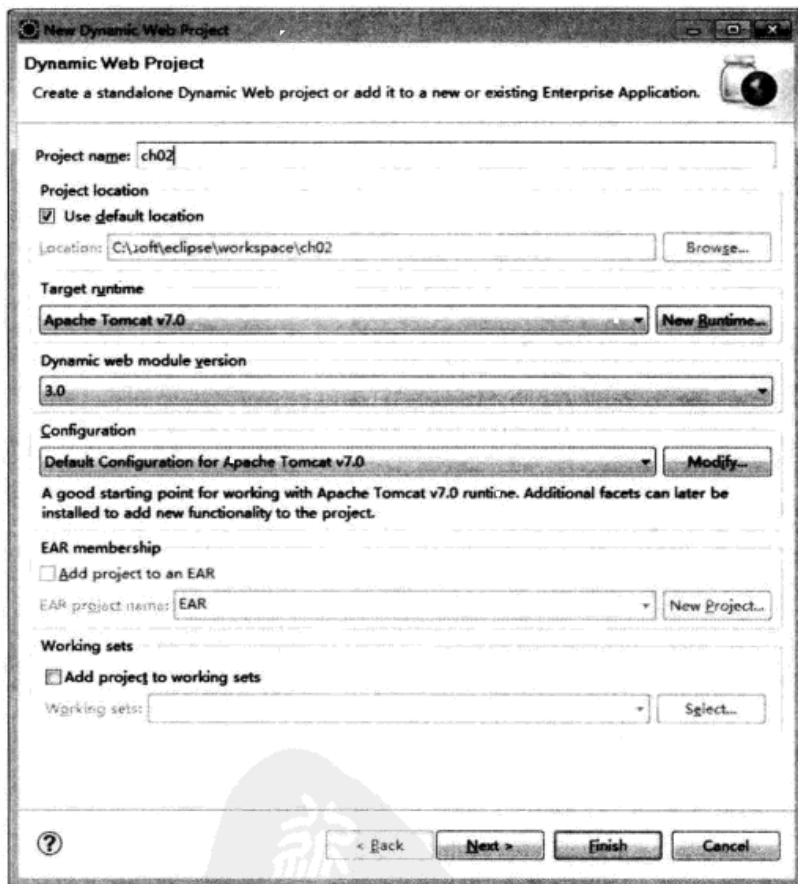


图 2-12 新建一个动态 Web 工程

**提示** 动态和静态这些概念初学者很难去体会，用 Flash 或 JavaScript 能够产生在网页中飞翔的图片，能产生一些网页中的动画特效，但这还不能称之为动态网页；运用了 JSP 后，网页的内容可以随着数据库的内容，在网页中的 Java 代码的协调之下动态地变幻网页中的数据内容，可称为 JSP 动态网页。

“Project name”后的文本框用于输入工程名称，读者可根据自己的需要取名，在调试程序时 Eclipse 就将这个名称作为 Web 应用的名称来部署。“Project location”用于设置工程所在的目录，如果“Use default location”前的复选框被选中了，则使用默认的路径，默认的路径为“Eclipse 工作区目录\工程名”，比如这里 Eclipse 的工作区目录为“c:\soft\eclipse\workspace”，当前工程名为“ch02”，则当前工程的默认路径为“d:\eclipse\workspace\ch02”。

“Target runtime”用于设置当前 Web 应用运行的目标 Web 容器，如 Tomcat 7，如果没有则可单击“New Runtime...”按钮配置一个新的。“Configuration”用于设置配置类型，采用默认的选项即可。

单击“Next>”按钮进入“Java”对话框，用于配置源代码文件夹，默认为 src 文件夹，保持默认即可。单击“Next>”按钮进入“Web Module”对话框，如图 2-13 所示。

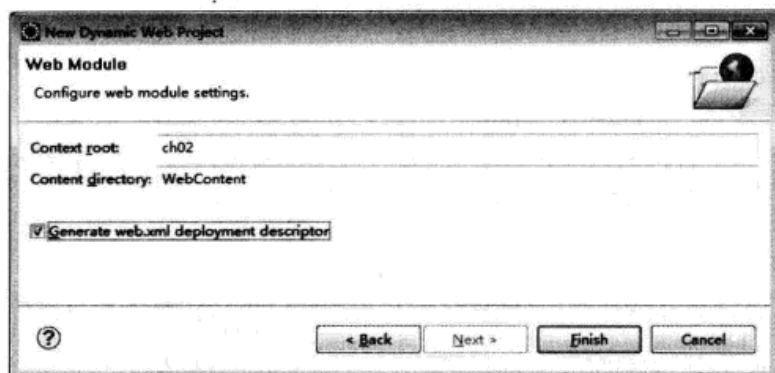


图 2-13 “Web Module”对话框

“Context root”指出上下文的根目录，即 Web 应用的根目录，“Content directory”指出 Web 应用内容目录，比如图 2-13 中为“WebContent”，则 Eclipse 会在当前 Web 工程中新建一个值为“WebContent”的文件夹，此外还会自动生成“WEB-INF”目录和 web.xml 文件。

“Generate web.xml deployment descriptor”表示是否生成发布脚本，建议选中此项。接下来单击“Finish”按钮完成开发框架的搭建。

搭建完成的 Web 系统框架如图 2-14 所示。可以看到 Eclipse 已经自动将 Tomcat 7 和 JDK 7 的 jar 文件放到了类库中（见 Libraries 菜单项）；已经自动生成了 Web 应用目录 WebContent，并生成了“WEB-INF”文件夹和 web.xml 配置文件，这些都是 Java Web 应用所必不可少的文件夹、配置文件。

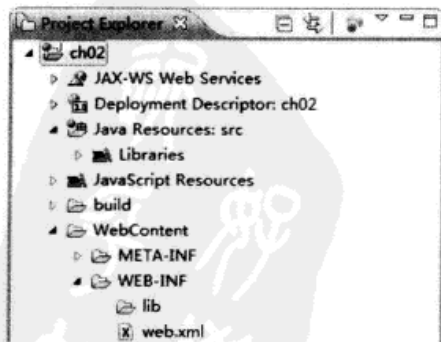


图 2-14 搭建完成的 Web 系统框架情况



### 2.3.2 开发一个 JSP 页面

JSP 页面用于在 Web 系统中展现数据。如图 2-15 所示的是 JSP 运行原理图，当 Web 服务器上的一个 JSP 页面第一次被请求执行时，JSP 引擎先将 JSP 页面文件转译成一个 Java 文件，即 Servlet。Java Servlet 是基于服务器端编程的 API，用 Java Servlet 编写的 Java 程序称为 servlet，servlet 通过 HTML 与客户交互。服务器将前面转译成的 Java 文件编译成字节码文件，再执行这个字节码文件来响应客户的请求。当这个 JSP 页面再次被请求时，将直接执行编译生成的字节码文件来响应，从而加快了执行的速度。

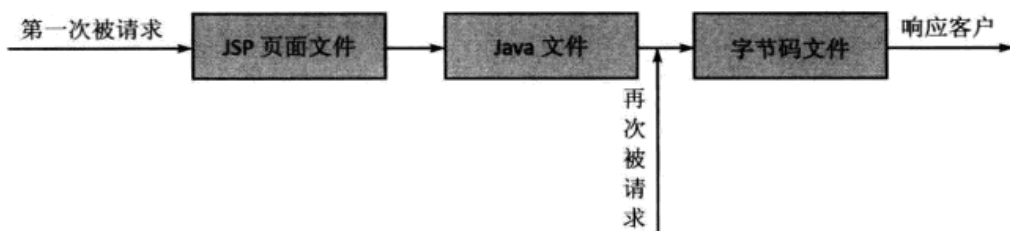


图 2-15 JSP 运行原理图

一般来讲，JSP 页面的开发过程是先由美工制作出网页，此时文件的扩展名为“.htm”或“.html”；再由程序员在网页中找到合适的位置插入 Java 代码，并将文件的扩展名更改为“.jsp”，这就成了 JSP 页面。

也有少数情况下，页面的美术效果是直接由程序员来制作的。术业有专攻，专业软件的开发，分工可能还会更细一些。因此我们的关注点应当放在如何在网页中编写程序而形成 JSP 页面上。下面就来看看如何用 Eclipse 开发 JSP 页面。

#### 【实例 2-1】新建第一个 JSP 页面

这里假定读者已经按“2.3.1”节中的内容搭建好了一个 Web 系统的框架，接下来看如何在这个 Web 系统中创建一个 JSP 页面。

在 Eclipse 左边的“Project Explorer”菜单中选中要新建 JSP 页面的文件夹，单击右键，在弹出的快捷菜单中选择“New”→“JSP File”菜单，进入“JSP”对话框，如图 2-16 所示。

在“File Name”后的输入框中输入要新建的 JSP 页面的文件名，注意输入文件名时要带扩展名.jsp。如果采用默认的 JSP 文件生成模板，在这个对话框中单击“Finish”按钮即可完成 JSP 页面的创建。此时，生成的 test.jsp 源代码如下。

test.jsp

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
  
```



```

</head>
<body>

</body>
</html>

```



图 2-16 新建一个 JSP 页面

自动生成的有些代码是多余的，可手工删除。此外，为支持中文的显示，需要将字符集设为 GBK 或 gb2312，修改后的源代码如下所示。

#### test.jsp

```

<%@ page language="java" contentType="text/html; charset=GBK"%>
<html>
<head>
<title>Insert title here</title>
</head>
<body>
这是我们编写的第一个 JSP 页面
</body>
</html>

```

接下来看如何运行这个 JSP 页面。在 Eclipse 中运行 JSP 有很多种操作方法，下面介绍其中的一种。

在 Eclipse 左边的“Project Explorer”菜单中选中 JSP 文件，单击右键，在弹出的快捷菜单中选择“Run As”→“Run on Server”，弹出“Run On Server”对话框，选择“Tomcat v7.0 Server”，单击“Next>”按钮，进入“Add and Remove”对话框，如图 2-17 所示。

在这个对话框中设置 Tomcat 启动时部署的 Web 应用。左边的“Available”列表中有当前 Eclipse 中可部署到 Tomcat 中的 Web 应用，右边的“Configured”中是准备部署的 Web 应用。

选中左边的 Web 应用后，单击“Add”按钮可将左边的 Web 应用转移到右边，也就是说，部署 Web 应用。单击“Remove”按钮可移除部署；单击“Add All”按钮可将左边的所有 Web 应用转移到右边，即部署所有的 Web 应用；单击“Remove All”按钮可将所有的 Web 应用部署移除。接着单击“Finish”按钮，Eclipse 即会自动启动 Tomcat，并在 Eclipse 中开启一个新的窗口，显示程序运行的结果，如图 2-18 所示。

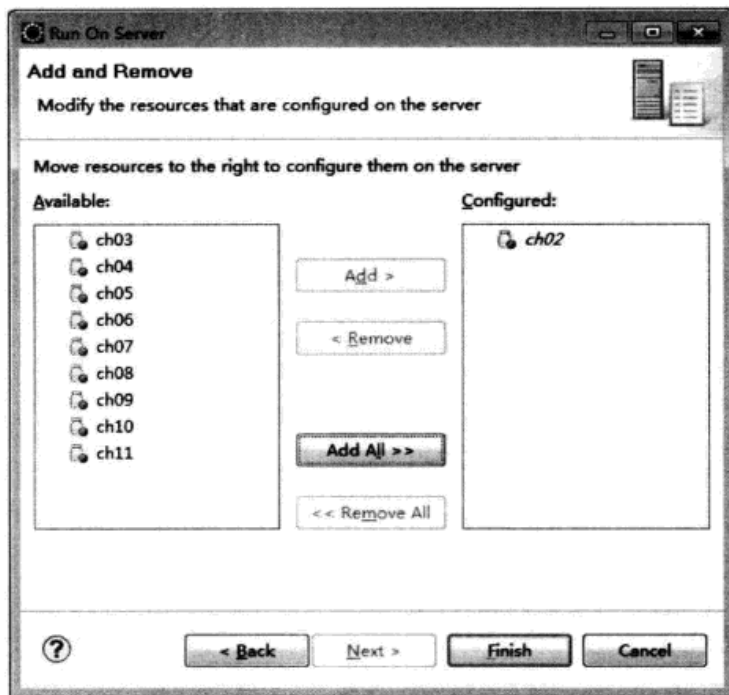


图 2-17 增加和删除工程

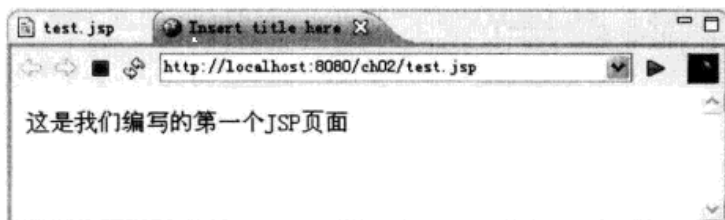


图 2-18 第一个 JSP 页面的运行结果

运行 JSP 页面时，Eclipse 的控制台会显示运行情况，如图 2-19 所示。

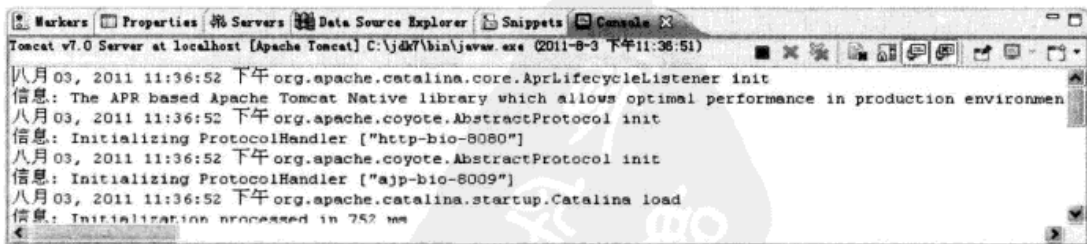


图 2-19 Eclipse 的控制台输出

如果其中有报错信息如下：



警告: [SetPropertiesRule]{Server/Service/Engine/Host/Context} Setting property 'source' to 'org.eclipse.jst.jee.server:... ' did not find a matching property.

则可以在“Servers”选项卡（如图 2-20 所示的 Eclipse 界面下半部分）中双击“Tomcat v7.0 Server at localhost”，会在 Eclipse 的编辑区出现“Tomcat v7.0 Server at localhost”对话框，在“Overview”视图下选中“Publish module contexts to separate XML files”前的复选框，再重新运行 JSP 文件就可以解决问题了。

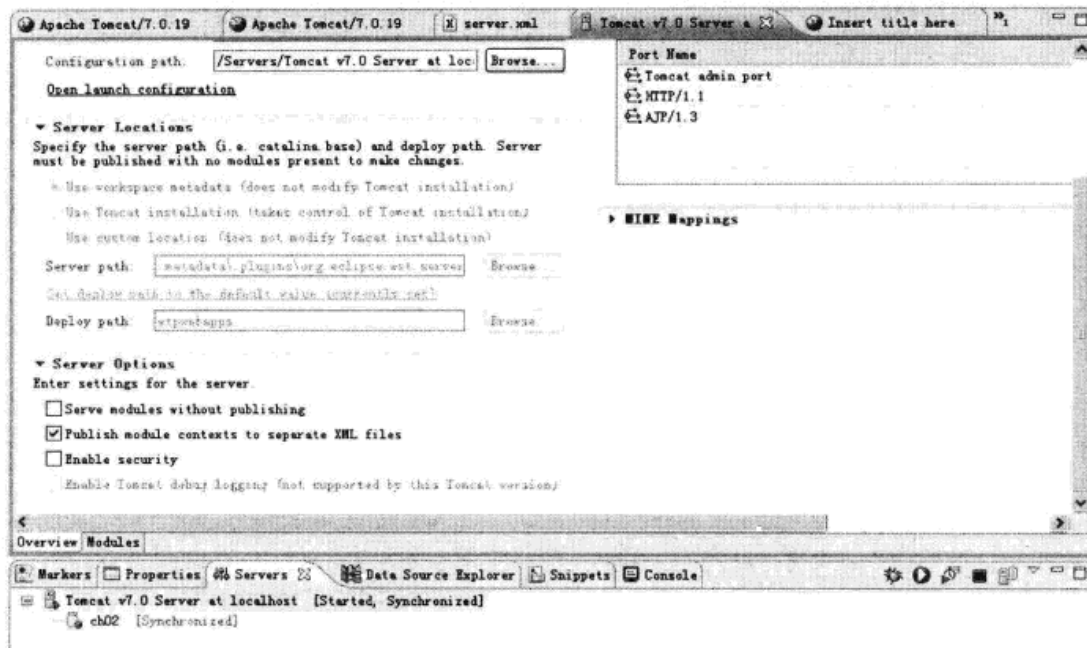


图 2-20 Eclipse 的操作界面

## 2.4 小结

在本章中，首先介绍了现在应用较为广泛的 3 种 JSP 服务器。Tomcat 由于占用系统资源少，而被较多地应用于程序编写与调试和中小型 Web 信息系统的开发中。

接着介绍了 JSP 运行环境的安装与配置，包括 JDK 7、Tomcat 7、Eclipse 的安装与配置，JDK 7 的配置需要增加系统变量并赋值。配置好开发环境后使用 Eclipse 搭建了一个 Web 系统框架，并开发了第一个 JSP 页面。

搭建起 Web 应用的开发框架是开发 Web 项目的基础性工作，无论是初学者还是开发行家都需要做这项工作内容，使用 Eclipse 将能事半功倍。搭建起 Web 应用的操作过程读者应多操作几遍，以后再新建工程就会很快了，因为操作已经熟悉，并且已经清楚哪些选项应当如何设置，也就熟能生巧了。利用这种向导式的模式来搭建框架也比手工搭建速度要快得多，而且相当准确。比如初学者在手工新建文件夹时就很容易在“WEB-INF”这个文件夹名上命名不规范而导致出错，因为 Java 对大小写敏感。

用 Eclipse 来编写 JSP 页面中的 Java 程序的好处有很多，比如 Eclipse 会自动与类库、自定义包中的类建立起关联关系，如果编写程序时出现无法找到类，或 Java 程序有语法错误，Eclipse



都能检测出来，此外还能提供方法的诱导输入功能，这样程序员就不必去记忆那些方法名称和参数类型、参数个数了，从而大大降低了手工编程时出错的可能性，在本书后续内容的学习中，读者还可以进一步领略。

## 2.5 练习

1. 安装并配置 JDK 7 与 Tomcat 7。
2. 在 Eclipse 中新建一个动态 Web 工程，然后创建一个 JSP 页面，显示“让我们一起来学习 JSP 吧！”

# 03

## Web 开发基础

---

编写 JSP 程序时，常常需要在 HTML 代码中合适的位置插入 Java 语句，因此要用 JSP 进行 Web 开发，程序员就要掌握基本的 HTML 知识。B/S 模式下的 Web 程序与客户的交互通常是通过客户端的 HTML 表单来提交信息，采用 JavaScript 对客户提交的信息数据进行验证，此外，采用 JavaScript 还可响应客户事件。

本章首先讲述 HTML 的基础知识，这些是每个从事 Web 开发的程序员所必须掌握的；接下来讲解基本的 JavaScript 内容，掌握 Web 信息交互的方法是最为重要的，也是初学 Web 编程的难点之处。

本章的内容比较简单，是 Web 开发的基础知识，旨在引导读者快速入门，对 HTML 及 JavaScript 比较了解的读者可跳过本章的内容，直接学习后面的章节。

### 3.1 HTML

#### 3.1.1 什么是 HTML

HTML (Hyper Text Markup Language, 超文本标记语言) 是一种描述网页的标记语言，用来描述如何将文本格式化。HTML 可以定义网页中文本和图形的格式，以及指向其他网页的链接。整个 Internet 就像一张巨大的网，一个 HTML 页面就像这张网中的一个点，另外，HTML 还指引浏览者从一个页面上通过单击超链接跳转到另一个 HTML 页面。

#### 3.1.2 URL

URL (Uniform Resource Locator, 统一资源定位器)，是 Internet 中用来唯一标识和定位 Internet 中资源的一种简单的命名机制。如下所示的 URL 由三部分构成：协议、主机 DNS 名和文件名。

`http://www.phei.com.cn/jc/rj.html`

在上面的 URL 中, 协议为“http”, 存放信息的主机为“www.phei.com.cn”, 文件名为“rj.html”, “jc/”是映射到用户网站服务器上的目录。

### 3.1.3 HTML 结构

HTML 标记(有时也称为标签)一般用尖括号“<”和“>”括起来, 中间为标记名称, 有开始标记和结束标记并配套使用(少数 HTML 标记只有一个标记), 如果要定义标记的属性则在标记名称后用空格和属性来定义内容。



**注意** HTML 对大小写并不敏感

一个 HTML 文档包括文档头和文档体两部分。文档头信息包含在标记<head>与</head>之间, 其中包含了一些有关此网页的信息, 如网页标题、导入样式表信息等。标记<body>与</body>中间是网页的文档体部分。其中标记<html>、<body>是必须要有的, 因为浏览器要根据此标记做初始化的工作。

如下面的 HTML 代码:

```
<HTML>
<head>
<title>HTML 网页</title>
</head>
<body>
这里是网页中的内容。
</body>
</HTML>
```

### 3.1.4 HTML 标记

这里仅介绍比较常用的 HTML 标记。

#### 1. 基本标记

(1) <html></html>标记, 向浏览器表明此文档为超文本文档, 文件以“.htm”或“.html”作为扩展名。

(2) <head></head>标记, 表示文档头, 包含一些初始化信息, 包括标题 title, JavaScript 及 meta 信息等。

(3) <body></body>标记, 表示文档体, 相当于 HTML 文档的正文部分。

(4) <title></title>标记中的信息将会显示在浏览器窗口的标题上。

(5) <p>标记是一个段落的开始标志。

(6) <hr>为水平线标记, 用于插入一条水平线。



**注意** <p>、<hr>无需和</p>、</hr>配对使用。



(7) `<img>` 标记，图形标记，主要用来将图片插入至网页中，格式如下：

```

```

`src="图片来源"`，表示图片来源，可接受 .gif、.jpg 及 .png 格式。若图片文件与该 HTML 文件同处一个目录则只需其文件名称，否则必须加上正确的途径，相对或绝对路径皆可。`width="宽度" height="高度"`，设定图片大小，此宽度及高度一般采用像素为单位。`hspace="图片左右边沿空白" vspace="图片上下边沿空白"`，设定图片边沿空白，以免文字或其他图片过于贴近，`border="图片边框厚度"`，设置图片边框厚度。`align="旁边文字的位置"`，用于调整图片旁边文字的位置，可选值：top, middle, bottom, left, right，默认为 bottom。`alt="图片文字"`，用以描述该图片的文字，若用户使用的是文字浏览器，由于不支持图片，这些文字便会代替图片而被显示。若用户使用的是支持图片显示的浏览器，当鼠标移至图片上时这些文字也会显示。`lowsrc="低解像图片"`，设定先显示低解像图片，若所加入的是一张很大的图片，下载时间会很长，这张低解像图片会先被显示。

(8) `<a></a>` 为超链接标记，格式如下：

```
<a href="目标 URL" target="结果窗口">超链接显示的文字内容</a>
```

`href="目标 URL"`，这个参数是链接目标的 URL，当作为一外部链接时，`href` 设定的是该链接的文件名称，若本文档与目标 URL 不是同在一目录下则须加上适当的路径，相对、绝对路径皆可。`target="结果窗口"`，设定单击超链接之后所要显示的视窗，可选值为 \_blank, \_parent, \_self, \_top，窗口名称。“\_blank”表示将链接的页面内容打开在新的浏览视窗中；“\_parent”表示打开该页面的页面，可以理解为当前文件的上一个页面；“\_self”表示将链接的页面内容显示在当前的视窗中；“\_top”表示将链接的页面内容显示在没有框架的视窗中。

## 2. 表格标记

(1) `<table></table>` 标记定义一个表格，其他表格标记只能在它的范围内才适用

(2) `<tr></tr>` 标记定义一个表格行 (row)

(3) `<td></td>` 标记定义一个单元格 (cell)。

在 Web 开发中，常用表格对网页元素进行定位，但又不显示出表格的单元格边框，因此网页将由一张张互相嵌套的边框宽度为 0 (`border="0"`) 的表格构成。

### 3.1.5 表单

(1) `<form></form>` 为表单标记，其他表单标记均需要在其内才有效。例如：

```
<form action="checklogin.jsp" method="POST">
```

表单通常和网页动态程序配合使用，以提交表单中的数据。参数 (有时也称为属性) `action` 用以指明处理该表单数据的程序所在的位置；参数 `method` 用于表明传送资料的方式，可选值常用的有 POST、GET。POST 方式允许传送大量资料，提交的信息不会显示在地址栏中，而 GET

方式只接受低于 1KB 的资料，提交的信息会显示在浏览器的地址栏中。

(2) `<input></input>` 为输入标记，例如：

```
<input type="text" name="num" value="20" align="middle" size="2"
maxlength="255">
```

参数 `type` 表示输入方式。输入方式为 `text`，能产生一个单行的文本输入框，上限为 255KB；`name` 为名称，`value` 为默认值，`align` 为对齐方式，`size` 为显示的长度，`maxlength` 为输入的字节数的上限。

```
<input type="radio" name="sex" value="female" align="middle" checked>
```

参数 `type` 为 `radio`，表示单选框，`checked` 表示在默认情况下该单选框被选中。

```
<input type="checkbox" name="hui" value="Leon" align="right" checked>
```

参数 `type` 为 `checkbox`，表示复选框，`checked` 表示在默认情况下该复选框被选中。

```
<input type="password" name="password1" value="999" align="MIDDLE" size="5"
maxlength="9">
```

参数 `type` 为 `password`，表示密码输入框，所输入的字符全以 “\*” 显示。

```
<input type="submit" name="tj" value="确定" align="midden">
```

参数 `type` 为 `submit`，表示表单提交按钮；当此参数值为 `reset` 时表示重置按钮，点击时会清除表单中输入的数据。

(3) `<select></select>`，用于表示选择下拉框。

(4) `<option></option>`，用于表示下拉框中的选项。

具体的表单如何与动态内容配合来完成客户信息的提交将在 3.3 节中做详细的介绍。

## 3.2 JavaScript

### 3.2.1 何谓 JavaScript

JavaScript 是一种脚本语言，结构简单，使用方便，其代码可以直接放入 HTML 文档中，也可以直接在支持 JavaScript 的浏览器中运行。JavaScript 使得网页的交互性更强，页面更生动和灵活。并且 JavaScript 所编写的程序可以捕获事件并对事件做出相应的反应。

### 3.2.2 加入 JavaScript

可以直接将 JavaScript 脚本加入 HTML 文档：

```
<script language="JavaScript">
JavaScript 语言代码
...
</script>
```

标记 `<script>...</script>` 指明 JavaScript 脚本源代码将放入其间，属性 `language="JavaScript"` 说明使用的是 JavaScript 语言。



### 3.2.3 JavaScript 对象

JavaScript 中已经预先定义了一些对象，以方便程序员使用。大多数预定义对象是 Navigator 对象的一部分，如图 3-1 所示的是其对象层次结构图。

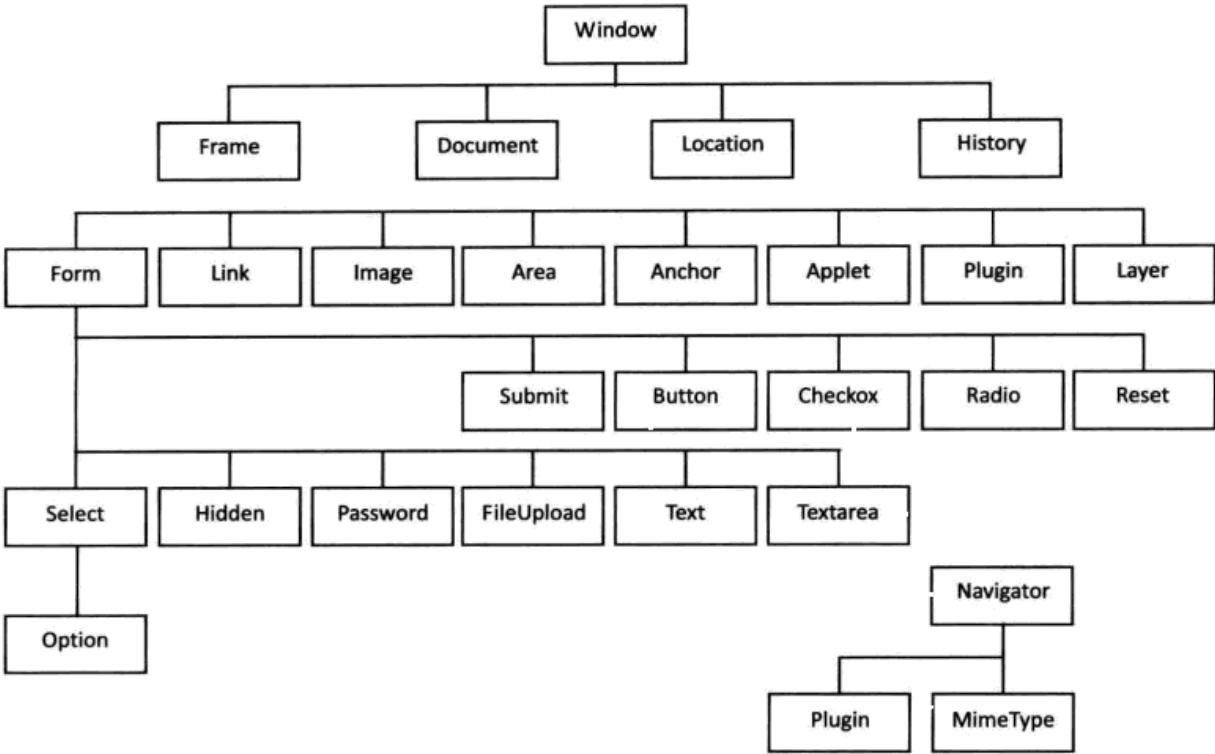


图 3-1 JavaScript 对象层次结构图

- Window: 处于对象层次的顶端，提供了处理 Navigator 窗口的方法和属性。
- Location: 提供与当前打开的 URL 一起工作的方法和属性。
- History: 提供与历史清单有关的信息。
- Document: 包含与文档元素（elements）一起工作的对象，它将这些元素封装起来供编程人员使用。

JavaScript 中的事件常用的有单击事件、改变事件和选中事件等，如表 3-1 所示。

表 3-1 JavaScript 中的事件

方法名	说明
单击事件 onClick	当用户单击鼠标按钮时，产生 onClick 事件
改变事件 onChange	在 text 或 textarea 中输入值改变时引发该事件，当在 select 中选项状态改变后也会引发该事件
获得焦点事件 onFocus	用户单击 Text 或 textarea 以及 select 对象时，产生该事件
选中事件 onSelect	当 Text 或 Textarea 对象中的文字被选中后，引发该事件
失去焦点 onBlur	当 text、textarea 及 select 对象失去焦点时引发该事件
载入文件 onLoad	当文档载入时，引发该事件
卸载文件 onUnload	当退出页面时引发该事件



### 3.3 Web 信息交互

#### 3.3.1 表单信息交互

在 HTML 中主要通过窗体对象（Form）与 JavaScript 进行信息交互。

窗体对象（Form）构成了 Web 页面的基本元素。通常一个 Web 页面可以有一个窗体或几个窗体，使用 Forms[ ]数组来实现不同窗体的访问，窗体对象最主要的功能就是能够直接访问 HTML 文档中的窗体。

窗体对象（Form）中的基本元素由按钮、单选按钮、复选按钮、提交按钮、重置按钮及文本框等组成，Form 元素的方法、事件与属性如表 3-2 所示。在 JavaScript 中要访问这些基本元素，必须通过对应特定窗体元素的数组下标或窗体元素名来实现。每一个元素都要通过该元素的属性或方法才能引用。

表 3-2 Form 元素的方法、事件与属性

text	属性	name	设定提交信息时的信息名称
		value	出现在窗口中对应 HTML 文档中 Value 的信息
		defaultvalue	Text 元素的默认值
	方法	blur()	失去焦点
		select()	选中文字
	事件	onfocus	获得焦点时产生
		onblur	失去焦点时产生
		onselect	文字被选中后产生
		onchange	值改变时产生
textarea			属性、方法、事件同 Text
select	属性	name	提交信息时的信息名称
		length	长度
		options	组成多个选项的数组
		selectedIndex	该下标指明一个选项
		text	某一选项的属性，选项对应的文字
		selected	某一选项的属性，当前选项是否被选中
		Index	某一选项的属性，当前选项的位置
		defaultselected	某一选项的属性，默认选项
	事件	onfocus	获得焦点时产生
		onblur	失去焦点时产生
		onchange	改变值时产生
button	属性	name	提交信息时的信息名称
		value	出现在窗口中对应 HTML 文档中 value 的信息
	方法	click()	单击按钮元素（button）时产生
	事件	onclick	单击按钮元素（button）时产生

续表

checkbox	属性	Name	提交信息时的信息名称
		value	出现在窗口中对应 HTML 文档中 value 的信息
		checked	checkbox 的状态 true/false
		defaultchecked	默认状态
	方法	click()	checkbox 的某一个选项被选中
	事件	onclick	checkbox 的选项被选中时产生
radio	属性	name	提交信息时的信息名称
		value	出现在窗口中对应 HTML 文档中 value 的信息
		length	单选按钮中的按钮数目
		defaultchecked	默认按钮
		checked	是否被选中
		index	选中按钮的位置
	方法	click()	checkbox 的某一个项被选中
	事件	onclick	checkbox 被选中时产生
hidden	属性	name	设定提交信息时的信息名称
		value	出现在窗口中对应 HTML 文档中 value 的信息
		defaultvalue	Text 元素的默认值
password	属性	name	设定提交信息时的信息名称
		value	出现在窗口中对应 HTML 文档中 value 的信息
		defaultvalue	Text 元素的默认值
	方法	select()	选择输入口令域
		blur()	丢失 password 输入焦点
		focus()	获得 password 输入焦点
submit	属性	name	提交信息时的信息名称
		value	出现在窗口中对应 HTML 文档中 value 的信息
	方法	click()	相当于单击 submit 按钮
	事件	onclick	单击该按钮时产生

下面来看一个 HTML 表单和 JavaScript 交互的实例。

### 【实例 3-1】HTML 与 JavaScript 交互实例

用户注册功能是许多网站经常要设计的，那么就以一个用户注册的表单为例，这里，JavaScript 主要用来验证数据的合法性，其表单的效果如图 3-2 所示。

调试时，因为 JavaScript 与 HTML 浏览器可以直接解析，所以没有将源文件放入服务器中调试。如果没有包含 Java 语句，完全可以在浏览器中直接调试，而不必将其放在 Web 服务器软件的目录下。



**注意** JSP 中的 Java 与 JavaScript 不同，JSP 中的 Java 是在服务器端运行的程序，编译后再执行，而 JavaScript 是在客户端运行的程序代码，由客户端浏览器解析执行。

新用户注册

用户名(\*):

密 码(\*):

再输一次密码(\*):

性别: ☒ 男 ☐ 女

出生年月:  年  月  日

电子邮件(\*):

家庭住址:

图 3-2 用户注册表单

源程序代码如下:

userRegist.html

```
<html>
<script language="javascript">
function on_submit(){//验证数据的合法性
    if (form1.username.value == ""){
        alert("用户名不能为空, 请输入用户名!");
        form1.username.focus();
        return false;
    }
    if (form1.userpassword.value == ""){
        alert("用户密码不能为空, 请输入密码!");
        form1.userpassword.focus();
        return false;
    }
    if (form1.reuserpassword.value == ""){
        alert("用户确认密码不能为空, 请输入密码!");
        form1.reuserpassword.focus();
        return false;
    }
    if (form1.userpassword.value != form1.reuserpassword.value){
        alert("密码与确认密码不同");
        form1.userpassword.focus();
        return false;
    }
    if (form1.email.value.length!= 0){
        for (i=0; i<form1.email.value.length; i++){
            if (form1.email.value.charAt(i)=="@") break;
            if (i==form1.email.value.length){
                alert("非法 E-Mail 地址!");
                form1.email.focus();
                return false;
            }
        }
    }else{
        alert("请输入 E-mail!");
        form1.email.focus();
    }
}
```



```

        return false;
    }
}
</script>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>新用户注册</title>
</head>
<body>
<form method="POST" action="regist.jsp" name="form1"
onsubmit="return on_submit()">
    新用户注册<br>
    用户名(*): <input type="text" name="username" size="20"><br>
    密 码(*): <input type="password" name="userpassword" size="20"> <br>
    再输一次密码(*): <input type="password" name="reuserpassword" size="20"> <br>
    性别: <input type="radio" value="男" checked name="sex">男
         <input type="radio" name="sex" value="女">女<br>
    出生年月: <input name="year" size="4" maxlength=4>年
               <select name="month">
                   <option value="1" selected>1</option>
                   <option value="2">2</option>
                   <option value="3">3</option>
                   <option value="4">4</option>
                   <option value="5">5</option>
                   <option value="6">6</option>
                   <option value="8">7</option>
                   <option value="9">9</option>
                   <option value="10">10</option>
                   <option value="11">11</option>
                   <option value="12">12</option>
               </select>月
               <input name="day" size="3" maxlength=4>日<br>
    电子邮件(*): <input name="E-mail" maxlength=28><br>
    家庭住址: <input type="text" name="address" size="20"><br>
    <input type="submit" value="提交" name="B1"><input type="reset"
    value="全部重写" name="B2"><br>
</form>
</body>
</html>

```

在 HTML 中定义了一个表单，名称为 form1，当表单提交时就调用 JavaScript 的函数 on\_submit() 来判断用户输入的数据是否合法，并根据此函数返回的数值决定是否提交表单数据。返回 true 表示数据合法，通过验证，可以提交表单；返回 false 表示数据非法，此时不能提交数据，请用户修改相关数据后再提交。

下面看在 on\_submit() 函数中如何判断用户输入的数据是否合法。

```

if (form1.username.value == ""){
    alert("用户名不能为空，请输入用户名！");
    form1.username.focus();
    return false;
}

```

在程序代码中，form1 是提交数据表单的名称。username 是表单用户名数据文本框的名称。





图 3-3 用户名为空时的报告对话框

如果 `username` 的值为空, 则弹出一个对话框, 如图 3-3 所示, 报告“用户名不能为空, 请输入用户名!”, 然后将输入焦点定位在此文本框上, 并返回 `false`, 表示存在非法数据, 表单数据不能提交, 需要用户进行修改。

同理, 输入的用户密码及确认密码的数据亦不能为空, 这里输入密码用的类型为 `password`, 在表单中输入密码时将会在浏览器中显示 “\*”, 以防止泄露密码数据。

```
if (form1.userpassword.value != form1.reuserpassword.value){
    alert("密码与确认密码不同");
    form1.userpassword.focus();
    return false;
}
```

这段代码用来判断两次输入的密码是否一致, 若不一致则要求用户修改, 直至两次输入的密码一致。要求用户两次输入密码是为了防止用户误输了密码。

```
if (form1.email.value.length!= 0){
    for (i=0; i<form1.email.value.length; i++){
        if (form1.email.value.charAt(i)=="@") break;
        if (i==form1.email.value.length){
            alert("非法 E-Mail 地址!");
            form1.email.focus();
            return false;
        }
    }
}else{
    alert("请输入 E-mail!");
    form1.email.focus();
    return false;
}
```

这段代码用来验证用户输入的 E-mail 地址是否合法, 首先不能为空, 即长度不为 0, 为 0 则报告给用户, 要求用户输入 E-mail 地址; 如果不为空, 则验证是否合法。这里用一个 `for` 循环检查输入的地址中是否含有 “@” 符号, 如果有则 “`i`” 变量记录了 “@” 符号所在位置的下标, 并终止循环, 然后比较 “`i`” 是否与字符串的长度相等, 因为如果地址中不含 “@” 符号, 两者必不相等, 据此来验证 E-mail 地址是否合法。

HTML 还可以用表单与 JSP 程序代码进行交互, 向服务器提交信息, 服务器再根据用户提交的信息做出响应, 主要使用 JSP 中的 `response`、`request` 等对象, 这些在后续的章节中讲解。

### 3.3.2 用正则表达式验证提交的数据

在进行 Web 信息交互的过程中, 经常会要求在 JavaScript 代码中验证数据的合法性, 比较常见的有验证数字、整数及电子邮件。验证的方法有两种, 一是如前面实例 3-1 所示, 编写一个定制的函数来实现; 二是利用正则表达式作为模式匹配。用正则表达式的方法, 只需要给出匹配串的正则表达式, 并调用相应的方法即可, 正则表达式就是用来描述模式匹配的规则。

什么是模式匹配？实际上就是指字符串的匹配问题，Web 信息交互的过程中其数据均是字符形式，模式则是指与源字符串进行匹配的字符串，可用正则表达式来表示。

正则表达式由两种字符构成。一种是元字符，元字符是指“\ | ( ) [ ] { } - ^ \$ \* ? . +”这些字符，用来限定一定的格式；除元字符以外的其他字符都是基本字符。在正则表达式中，元字符不能直接作为模式的一部分，需要进行转义，转义的方法是在元字符前面加上反斜杠“\”，如正则表达式“\ (deng\ )”表示模式字符串为“(deng)”。



**注意** 元字符都是半角的符号。

“.”表示任意一个除换行符以外的字符，如“d.b”表示dab、dbb、dcb等字符串。“|”表示或者的意思，如“a|b”，表示可以是a或者b。一对中括号“[]”结合起来使用可表示某特定类型的字符，如“[dzy]”表示可以是d、z、y这三个字符中的任意一个，它等价于正则表达式“d|z|y”。“-”表示一段字符的范围，如“[A-Z]”表示可以是所有的大写字母。“\$”表示模式必须出现在目标串的结尾，如“un\$”表示以un结尾的字符串，可以是aun、bun等。“^”表示除了指定类型以外的字符，如“[^a-z]”表示除小写字母以外的任意一个字符。



**注意** 在正则表达中，如果把“^”放在模式的最前面（不能是类型的里面，如上段例子中“[]”的里面），表示头部必须匹配。

括号“()”结合使用表示一个字模式。“?\*+”是数量限定符，“?”表示0个或1个，“\*”表示0个或多个，“+”表示1个或多个，如“ad?”可匹配a、ad、add；“ad\*”可匹配a、ad、add、addd等；“ad+”可匹配ad、add、addd等。

大括号“{}”结合使用表示匹配的次数。形如“{n}”表示匹配确定的n次，n是一个非负整数，如“de{2}”可匹配dee。形如“{n,}”表示匹配至少n次，n是一个非负整数，如“de{2,}”可匹配dee、deee等。形如“{n,m}”表示匹配至少n次，最多m次，n与m均为非负整数，且n≤m，如“ad{2,4}”可以是add、addd、addddd。

正则表达式中还有一类特殊的用法，就是预定义词。预定义词如表3-3所示。

表 3-3 正则表达式中的预定义词

预定义词	含 义
\d	一个数字，同[0-9]
\D	一个非数字字符，同[^0-9]
\s	一个白字符，同[\t\n\x0B\f\r]（注意其中包括空格）
\S	一个非空白字符同[^ \t\n\x0B\f\r]或[^ \s]
\w	一个字符，可以是字母、数字或下划线，同[a-zA-Z_0-9]
\W	一个字符，不能是字母、数字或下划线，同[^ \w]

表 3-4 给出了一些常用的模式正则表达式。



表 3-4 一些常用的模式正则表达式

类 型	正则表达式	模式含义
数学类	<code>^\d+(\.\d+)*\$</code>	数字
	<code>^\d+\$</code>	非负整数（正整数或0）
	<code>^[0-9]*[1-9][0-9]*\$</code>	正整数
	<code>^((-)\d+)((0+))\$</code>	非正整数（负整数或0）
	<code>^-([0-9]*[1-9][0-9]*)\$</code>	负整数
	<code>^-?\d+\$</code>	整数
	<code>^\d+(\.\d+)?\$</code>	非负浮点数（正浮点数或0）
	<code>^(((0-9)+\.[0-9]*[1-9][0-9]*) ((0-9)*[1-9][0-9]*\.[0-9]+) ((0-9)*[1-9][0-9]*))\$</code>	正浮点数
	<code>^((-)\d+(\.\d+)?)((0+(\.0+)?))\$</code>	非正浮点数（负浮点数或0）
	<code>^-(((0-9)+\.[0-9]*[1-9][0-9]*) ((0-9)*[1-9][0-9]*\.[0-9]+) ((0-9)*[1-9][0-9]*))\$</code>	负浮点数
字符类	<code>[\u4e00-\u9fa5]</code>	中文字符
	<code>[\x00-\xff]</code>	双字节字符(包括汉字)
字符串类	<code>^[A-Za-z]+\$</code>	由26个英文字母组成的字符串
	<code>^[A-Z]+\$</code>	由26个大写英文字母组成的字符串
	<code>^[a-z]+\$</code>	由26个小写英文字母组成的字符串
	<code>^[A-Za-z0-9]+\$</code>	由数字和26个英文字母组成的字符串
	<code>^\w+\$</code>	由数字、26个英文字母或者下划线组成的字符串
	<code>^\w-+([\w-]+)*@([\w-]+([\w-]+)+)\$</code>	E-mail 地址
	<code>\n[\s  ]*\r</code>	空行
	<code>/&lt;(.*?)&gt;.*&lt;/1&gt; &lt;(.*?) /&gt;/</code>	HTML 标记
	<code>(^s*) (\s*\$)</code>	首尾空格
	<code>^[a-zA-Z]+://(\w+(-\w+)*)(\.( \w+(-\w+)*))*(\? \S*)?\$</code>	URL

## 【实例 3-2】正则表达式验证数据实例

这个实例演示了如何在 Web 交互中用 JavaScript 的正则表达式来验证用户输入的数字、整数和电子邮件地址是否合法。源代码如下：

## checkData.jsp

```

<%@ page contentType="text/html; charset=gb2312" %>
<html>
<head>
<title>正则表达式验证示例</title>
<script language="JavaScript">
<!--
function checkdata() { //检查函数
//检查是否为数字
var txt = document.forms[0].num.value;
if(txt.search("^\d+(\.\d+)*$")!=0) {

```

[illegible]

程序运行结果如图 3-4 所示。



图 3-4 正则表达式验证示例

当输入不合法时，会弹出警告信息，并要求用户继续输入；当输入均合法时，则弹出检验通过信息对话框。



程序中用 `checkdata()` 函数检查这三个输入框中输入的数据是否合法。函数中用到了 JavaScript 字符串对象的 `search` 方法，对于 `search` 方法如果匹配成功，则返回模式在目标串中的位置，若失败则返回 -1。上述用的正则表达式均为整串匹配，因此匹配成功时应返回 0。



**注意** 方法中的 “\” 需要转义，故为 “\\”。

模式匹配在 Java 程序中也经常会被用到，这是因为仅仅做客户端脚本验证是不够的，还需要做服务器验证，为什么呢？因为有的浏览器可以禁用脚本执行，也可以直接通过 URL 将表单数据提交到目标页面，从而绕过 JavaScript 脚本的验证，但会带来安全隐患。Java 程序中用字符串对象的 `matches` 方法作为模式匹配。在后续章节中会出现客户端和服务端同时需要验证的实例。

## 3.4 小结

HTML 是 Internet 中表述文档的一种描述性语言，它描述了文本、图片等元素的格式，并用超链接把 HTML 网页链接起来形成一张巨大的信息网。JavaScript 是由客户端浏览器负责解析执行的一种脚本语言，它可以调用 HTML 对象中的元素，一般用来做数据验证以及与服务端无关的事件。

Web 信息的交互也可以在 HTML 元素与服务端端的 JSP 程序之间发生，常以提交表单的形式来传递用户输入的数据。

正则表达式可表示模式匹配中的模式串，以简化数据验证规则程序的编写。为安全起见，在 JavaScript 验证数据合法性规则时，同样需要在服务器端应用做出验证。

## 3.5 练习

1. 编写一个 HTML 表单，并用 JavaScript 验证其提交数据的合法性。（注：可以参考实例 3-1）
2. 编写一个 HTML 表单，其中有用来输入 E-mail 地址、联系电话、姓名（必须输入汉字）等字段的位置，并用正则表达式的方法来验证数据的合法性。（注：程序可以参考实例 3-2，模式参见表 3-4）





# 04

## JSP 语法

Java 语言的语法与 C 语言类似，因此学习过 C 语言的用户将可以更快上手开发 Java 程序。学习一种语言的语法是程序员利用这种语言进行开发的基础，JSP 语法是建立在 Java 语言基础上的，但由于它是一种 Web 程序设计语言，故有一些自己特有的用法和指令。

本章将首先讲述 JSP 网页程序的基本结构；然后讲解变量、方法和类的声明，指明它是一种面向对象的程序设计语言，具有良好的可复用性；接着介绍在 JSP 开发中经常要用到的 JSP 指令和动作指令。

通过本章的学习，读者应当对 JSP 的语法有较为深刻的认识，并能灵活运用它们编写简单的 JSP 程序。

### 4.1 JSP 的基本结构

在 HTML 页面中插入 Java 程序即成为 JSP 页面程序，在 JSP 中包括两种主要内容，一是 HTML、JavaScript 语言，是静态的内容，由客户端浏览器负责解释执行；二是 Java 程序及其相关元素，是动态的内容。Java 程序及其相关元素包括 Java 程序片、表达式、JSP 指令与动作标签以及 Java 变量、方法和类的声明。动态的内容将由服务器负责执行，将结果转变为字符串，然后把结果和静态内容一起交给客户端的浏览器显示出来。下面来看一个程序实例。

#### 【实例 4-1】JSP 程序的基本结构

instring.jsp

```
<%@ page contentType="text/html; charset=gb2312"%><!--JSP 指令标签-->
<%@ page import="java.util.*"%>                                <!--JSP 指令标签-->
<html><!--HTML 标记符-->
<body>
<form name="form1" action="instring.jsp" method="post">
<input type="text" name="jcs">
<input type="submit" name="submit" value="提交">
```

```

</form>
<%!String s=null;%><!--变量声明-->
<%//以下为Java 程序片
    s=request.getParameter("jcs");
    if(s==null){
        out.println("您还没有输入,请输入!");
    }else if(s==""){
        out.println("您输入的是空字符串!");
    }else{
        out.println("您输入的是: "+s);
    }
%>
</body><!--HTML 标记符-->
</html>

```

在上述代码中,第1、2行为JSP指令标签;第3~7行为HTML标记符,在这几行中声明了一个表单用来提交数据,表单名称为form1,提交方法为post,提交给instring.jsp页面本身,表单中声明了一个文本框,名称为jcs,又声明了一个提交按钮;接下来声明了一个字符串变量s,初始值为null;在Java程序片中,用if语句进行了判断,当提交的文本框内容为空串(即长度为0的字符串)时向浏览器报告“您输入的是空字符串!”,当不为空时,向浏览器报告文本框中所提交的字符串。

程序的运行结果如图4-1所示。



图4-1 实例4-1的运行结果

可见要学好JSP程序开发,还要学好Java的数据类型、运算符与表达式、程序控制逻辑等Java的基本语法知识。

**注意** 字符串为null与为空串是不同的,为null表示变量还没有在内存中分配空间,空串则已经分配了空间。本例中第一次打开页面时为null,提示信息为“您还没有输入,请输入!”,如果没有输入而提交表单,则会提示“您输入的是空串!”,因为提交表单时,把文本框当做一个长度为0的字符串提交给了服务器端,此时服务器端用request的getParameter()方法接收该变量时为其分配了内存空间。

## 4.2 数据类型

Java中的数据类型可分为两种:简单数据类型和复合数据类型。简单数据类型包括整数、

实数、布尔数及字符等，复合数据类型由简单数据类型组合而成，如类、接口等。

在 Java 语言中，不支持指针类型数据、结构体类型数据、枚举型数据以及联合型数据，这种数据类型在 C/C++ 中常被用到，由 C/C++ 转为学习 Java 的读者请特别注意这一点；而且在 Java 中，字符串不是作为数组来处理的，而是一种对象，用类 `String` 或类 `StringBuffer` 可以定义一个字符串对象。

### 4.2.1 数据类型概述

Java 的数据类型如表 4-1 所示。

表 4-1 Java 的数据类型

数据类型	数据细分类型	占用空间
简单数据类型	布尔型 <code>boolean</code>	1 位
	字符型 <code>char</code>	2 字节
	字节型 <code>byte</code>	1 字节
	短整型 <code>short</code>	2 字节
	整型 <code>int</code>	4 字节
	长整型 <code>long</code>	8 字节
	单精度型 <code>float</code>	4 字节
	双精度型 <code>double</code>	8 字节
复合数据类型	类 <code>class</code>	
	接口 <code>interface</code>	
	数组	

在上表中，`short` 类型很少使用，因为这种类型的数据限制了存储时先高字节后低字节，致使在有的计算机中会出错。可以看出，Java 中的所有数据类型的长度都是固定的，与平台无关。

### 4.2.2 标识符

标识符用来标识变量、类、方法、对象、接口和包，Java 中标识符的名称必须符合以下的规则：

- (1) 必须以字母、下画线 “\_” 或美元符 “\$” 开头；
- (2) 标识符中除首字符以外的其他字符可以由字母、数字、下划线及美元符组成，其中不能有空格和连字符 “-”；
- (3) 标识符不能是关键字；
- (4) 大小写敏感。

Java 中默认情况下使用的是 Unicode 国际标准字符集，这样程序员可以使用汉语、法语等作为变量名，但提倡使用字母作为标识符。

下列是 Java 的关键字，读者在编写程序给标识符命名时注意不要使用。

`abstract`; `assert`; `boolean`; `break`; `byte`; `case`; `catch`; `char`; `class`; `const`; `continue`; `default`;



do; double; else; extends; final; finally; float; for; goto; if; implements; import; instanceof; int; interface; long; native; new; package; private; protected; public; return; short; static; strictfp; super; switch; synchronized; this; throw; throws; transient; try; void; volatile; while。

true、false 分别表示真、假的布尔值，null（空值）虽然不是关键字，但也不能作为标识符名称使用。

### 4.2.3 简单数据类型

#### 1. 整型数据

Java 中的整型数据可以是十进制、八进制或十六进制数值。十进制数值与日常的表达方式一样。八进制数值以 0 开头，如 022 表示的八进制对应的十进制数值为 18，-022 表示的八进制对应的十进制数值为-18。十六进制以 0x 或 0X 开头，如 0x22 表示的十六进制数对应的十进制数值为 38，-0X22 表示的十六进制对应的十进制数值为-38。

整型数据类型包括 byte、short、int、long，其中 short 类型使用较少，int 较为常用，在内存中占据 32 位空间，但实际应用中经常会超出这一范围，这时就得使用 long 类型（长整型），在数据后加 L（或 l）就表示为长整型，它在内存中占据 64 位空间，如 22L。

#### 2. 浮点型数据

浮点型数据中必须有小数点，有十进制和科学计数法两种形式。十进制表示法如 100.0、.231、231.；科学计数法表示如 231e12、231E12。

浮点型数据又分为单精度（float）和双精度（double）两类。其 double 占用 8 字节的空间，比 float 有更大的表示范围。

#### 3. 字符型数据

字符型常量指以单引号 “'” 括起来的字符，如 '3'、'C'。变量的类型是 char，占用 2 字节的空间。



**注意** Java 中字符型数据不是整数，这与 C/C++ 中不同，但可以作为整型数据来操作。

如下所示的代码：

```
char two='2';
int four=4;
char six=(char)(two+four);
```

第 1 行程序声明了一个字符常量 two，值为 '2'；第 2 行声明了一个整型常量，值为 4；第 3 行将两者相加，将结果强制转换为字符型数据并赋给 six。因为字符型数据可以作为整数来操作，所以 six 的值为 '6'。

Java 中支持转义字符，用反斜杠（\）开头，表示一些特殊字符，常用的转义字符及其含义如下所示。

\ddd: 表示 1~3 位八进制数值所表示的字符, 用 ASCII 码表示;

\uxxxx: 表示 1~4 位十六进制数值所表示的字符, 用 ASCII 码表示;

\': 单引号;

\\: 反斜杠;

\r: 回车;

\n: 换行;

\f: 走纸换页;

\t: 横向跳格;

\b: 退格。

#### 4. 布尔型数据

布尔型数据有两个值 `true` 和 `false`, 与 C/C++ 中不同, 它不对应整型数据, 常用在判断的场合, 如程序流控制中。

#### 【实例 4-2】简单数据类型综合应用实例

allexample.jsp

```
<%@ page contentType="text/html;charset=gb2312"%><!--JSP 指令标签-->
<%@ page import="java.util.*"%>                                <!--JSP 指令标签-->
<html><!--HTML 标记符-->
<body>
<%//以下为 Java 程序片
    boolean booleanTemp=true;
    byte byteTemp=80;
    char charTemp='c';
    int intTemp=234;
    short shortTemp=235;
    long longTemp=1234567;
    float floatTemp=1.234F;
    double doubleTemp=1.23E-8;
    out.println("布尔变量 booleanTemp 值为: "+booleanTemp+"<br>");
    out.println("字符型变量 charTemp 值为: "+charTemp+"<br>");
    out.println("整型变量 intTemp 值为: "+intTemp+"<br>");
    out.println("短整型变量 shortTemp 值为: "+shortTemp+"<br>");
    out.println("字节型变量 byteTemp 值为: "+byteTemp+"<br>");
    out.println("长整型变量 longTemp 值为: "+longTemp+"<br>");
    out.println("单精度型变量 floatTemp 值为: "+floatTemp+"<br>");
    out.println("双精度型变量 doubleTemp 值为: "+doubleTemp+"<br>");
%>
</body><!--HTML 标记符-->
</html>
```

在上述程序代码中, 从第 5 行开始为 Java 程序片, 声明了几种简单的数据类型, 然后在浏览器中输出这些数据类型变量中的值。

程序运行的结果如图 4-2 所示。





图 4-2 实例 4-2 简单数据类型综合应用实例程序的运行结果

## 5. 简单数据类型之间的转换

简单数据类型之间的转换在编程工作中经常会用到，并且很容易出错。在 Java 中，整型、浮点型和字符型数据被认为是同一类数据，其优先关系如下所示：

byte→short→char→int→long→float→double  
低                        高

在进行运算时，编译器会自动把优先级低的变量的数据类型转换为一起运算的更高优先级变量的数据类型，但优先级高的变量的数据类型转换为优先级低的变量的数据类型时必须进行强制转换，这样常常会降低数据的精度，使用时一定要注意。

进行强制转换的方法如下所示:

```
float floatTemp;  
int intTemp=(int)floatTemp;
```

从上述代码的第 2 行可以看出，在变量前加上一个要转换成的数据类型并用括号括起来，即可进行强制转换运算。

## 6. 简单类型的包装类

在 Java 中，可直接把简单数据类型的变量表示为一个类，以方便将简单类型的数据作为一个对象来处理。简单数据类型对应的包装类共有 6 个，分别是 Boolean、Character、Integer、Long、Float、Double。

有了包装类，就可以利用它们的构造函数来进行赋值操作，其构造函数如下所示。

```
Boolean(boolean)
Character(char)
Integer(int)
Long(long)
Float(float)
Double(double)
```

因为它们是类，所以声明时需要使用 `new` 关键字来给其分配内存空间，如：

```
boolean bTest=true;
Boolean objTest=new Boolean(bTest);
```

获取包装类的值时，要用形如 “\*\*Value()” 的方法，如 Boolean 类则为 booleanValue()。



## 【实例 4-3】包装类综合应用实例

bzexample.jsp

```

<%@ page contentType="text/html; charset=gb2312"%><!--JSP 指令标签-->
<%@ page import="java.util.*"%> <!--JSP 指令标签-->
<html><!--HTML 标记符-->
<body>
<%//以下为 Java 程序片
    boolean booleanTemp=true;
    Boolean BooleanTemp=new Boolean(false);
    byte byteTemp=80;
    Byte ByteTemp=new Byte(byteTemp);
    char charTemp='c';
    Character CharacterTemp=new Character(charTemp);
    int intTemp=234;
    Integer IntegerTemp=new Integer(intTemp);
    short shortTemp=235;
    Short ShortTemp=new Short(shortTemp);
    long longTemp=1234567;
    Long LongTemp=new Long(longTemp);
    float floatTemp=1.234F;
    Float FloatTemp=new Float(floatTemp);
    double doubleTemp=1.23E-8;
    Double DoubleTemp=new Double(doubleTemp);
    out.println("布尔型包装对象 BooleanTemp 值为: "+BooleanTemp.booleanValue()+"<br>");
    out.println("字符型包装对象 CharacterTemp 值为: "+CharacterTemp.charValue()+"<br>");
    out.println("整型包装对象 IntegerTemp 值为: "+IntegerTemp.intValue()+"<br>");
    out.println("短整型包装对象 ShortTemp 值为: "+ShortTemp.shortValue()+"<br>");
    out.println("字节型包装对象 ByteTemp 值为: "+ByteTemp.byteValue()+"<br>");
    out.println("长整型包装对象 LongTemp 值为: "+LongTemp.longValue()+"<br>");
    out.println("单精度型包装对象 FloatTemp 值为: "+FloatTemp.floatValue()+"<br>");
    out.println("双精度型包装对象 doubleTemp 值为: "+doubleTemp+"<br>");
%>
</body><!--HTML 标记符-->
</html>

```

程序运行的结果如图 4-3 所示。



图 4-3 实例 4-3 包装类综合应用实例程序的运行结果

## 4.2.4 数组

数组是一种复杂的数据类型，它是一组相同数据类型元素的集合，分为一维数组、二维数组和多维数组，通过数组名和下标可以唯一确定数组中的元素。

### 1. 一维数组

一维数组定义的语法格式如下：

```
数据类型 数组名[ ]
```

数据类型可以是简单的数据类型，也可以是复杂的数据类型，数组名称可以是任意一个合法的标识符。例如，下面的语句定义了一个整型数组。

```
int intArray[ ];
```



**注意** 在 Java 中定义了数组但并不为其分配内存空间，这与 C/C++ 不同，如果要在声明时就分配内存则应使用 `new` 关键字，例如：

```
数据类型 数组名[] = new 数据类型[数组长度]
```

用以上的格式声明数组的同时也分配了内存，例如：

```
int intArray[] = new int[10];
```

这样就为 `intArray` 分配了内存空间，并将其中的元素初始化为 0。声明数组时也可以把变量类型声明与分配空间结合起来。以下的两种声明语句是等效的：

```
int intArray[] = new int[10];
```

或

```
int[] intArray = new int[10];
```

声明数组时还可以同时进行数据的初始化工作，把要初始化的数据放在方括号内，用逗号隔开，数据可以是表达式，也可以是直接的数据。Java 会给声明的数组自动分配足够的内存空间，此时不必用 `new` 运算符，如下面的语句：

```
int intArray[ ] = {1,2,1,3,4,7,6};
```

### 2. 二维与多维数组

二维数组是在一维数组的基础上多一个下标，类似于数学中的矩阵，标识数组中唯一的元素就需要有两个下标，多维数组则需要有相应个数的下标。它的声明方法与一维数组基本相同，只是在声明时多用一个方括号，初始化时需要有相应层次的花括号。来看下面声明二维数组的程序代码：

```
int intArray1[][] = new int[10][9];
```

```
int intArray2[][] = {{1,2,3},{2,3,4},{3,4,5}};
```

第一句代码声明了一个 10 行 9 列的二维整型数组，并分配空间，所有的数据被初始化为 0；第二句代码声明了一个 3 行 3 列的二维数组，并同时给每个数据元素赋初始值。



**【实例 4-4】数组应用实例**

intArray.jsp

```
<%@ page contentType="text/html; charset=gb2312"%>
<html>
<body>
<%
    int i;//一维数组下标
    int j;//二维数组下标
    int k;//计数变量
    int intArray[][]=new int [5][6]; //声明一个 5 行 6 列的二维数组
    k=0;
    //数组赋初始值
    for(i=0;i<5;i++){
        for(j=0;j<6;j++){
            intArray[i][j]=k;
            k++;
        }
    }
    //输出数组中的值
    for(i=0;i<5;i++){
        for(j=0;j<6;j++){
            out.print(intArray[i][j]+" ");
            out.print("<br>");
        }
    }
%>
</body>
</html>
```

上述代码首先声明了一个 5 行 6 列的整型数组，然后用一个双重循环将数组中的数据初始化，再用一个双重循环输出数组所有元素的值。程序运行的结果如图 4-4 所示。

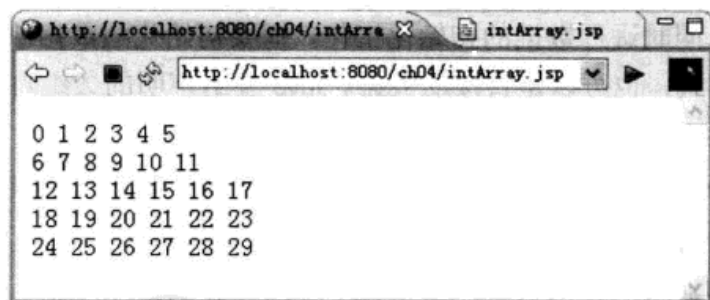


图 4-4 实例 4-4 数组应用实例程序的运行结果

## 4.2.5 类

类是一种复杂的数据类型。Java 是一种面向对象的程序设计语言，该语言的核心是支持类技术。要成为一名优秀的 Java 程序员，在很大程度上取决于对类技术理解和运用的程度。在 JDK 的类库中提供了许多标准的类，读者编写时可以直接拿来使用，不过也经常需要自行定义满足应用所需的类。

下面一起来学习类的方法、构造函数、this 以及设计自己的类等基础知识，并学会用类来声明对象。

## 1. 类的基础知识

可以把类理解为一种新的数据类型，一旦声明后，就可以用它来创建对象了。类就是对象的抽象，对象是类的实例，所以要使用对象，就必须先定义类。

类包括了数据和代码两部分。数据（也称为属性）是类中的实例变量，代码是类中的方法，两者都称为类的成员。

如下是简单的类声明及对象声明的程序代码。

```
class Box { //定义一个类
    public double length;
    public double width;
    public double height;
}
Box thisBox=new Box(); //声明一个类的实例
```

以上代码首先定义了一个类 Box，即盒子，它有三个数据：length（长）、width（宽）、height（高），但定义后它并不会生成实际的对象实例。接下来声明了这个类的一个实例，即对象，对象的名称为 thisBox，这个对象中会有类中的每个数据的复制。



**注意** 在 Java 中所有的对象空间都是动态分配的，所以声明对象时要用 new 运算。

声明了对象后，要使用类中的数据或方法则用“.”运算符，左边是对象名，右边是数据或方法名，如果要把 thisBox 的 length 数据设置为 120，则用如下语句：

```
thisBox.length=120;
```



**注意** 声明为 public 的数据可以直接赋值，但 private 的数据则应当使用类的方法来赋值和得到值，一般使用与数据同名的前加“set-”的“setXxxx()”方法，比如 length 属性使用以下方法赋值和得到值。

```
public void setLength(double length){ //赋值的方法
    this.length=length;
}
public double getLength(){ //得到值的方法
    return this.length;
}
```

## 2. 变量赋值

来看如下的两个程序代码。

```
Box box1=new Box();
Box box2=box1;
```

第一句程序代码声明了一个对象实例 box1，第二句在两个对象之间有“=”运算符。如果是简单数据类型，则它们之间是数据的复制，但如果是对象则两者之间将是一种引用的关系，



也就是说当 `box1` 中的数据改变了, 那么 `box2` 中的数据也会改变, 因为它们其实是同一个对象。

### 3. 类的方法

类的方法用来实现类所需要的一些功能。定义方法的格式如下:

```
限制类型 返回的数据类型 方法名称(参数列表){
    //方法体, 在这里添加程序代码
}
```



**注意** 在方法参数的传递中, 如果传递的是简单数据类型, 则传递的是数据的复制, 如果传递的是一个对象, 则传递的是这个对象的引用。

限制类型有 `public`、`protect`、`private`, 默认情况下为 `private`, 即私有的, 此时只有该类的其他方法能访问该方法; `public` 表示公有的, 即访问不受限制。

返回的数据类型指明了方法返回数据的数据类型, 如果不返回任何数据则类型为 `void`。在方法体中, 要返回值需要使用 `return` 语句, 这与 C/C++ 相同; 参数列表是传递给方法的参数列表; 花括号中间为方法体, 即实现方法的程序代码。

下面在 `Box` 类中加入一个求盒子容积的方法, 加入后 `Box` 类的程序代码如下所示。

```
class Box { //定义一个类
    public double length;
    public double width;
    public double height;
    public double boxVolume(){
        double volumeTemp;
        volumeTemp=length*height*width;
        return volumeTemp;
    }
}
```

代码中的 `boxVolume()` 方法用来求出盒子的容积, 并返回。如果声明了 `Box` 类的一个对象 `box1`, 要得到 `box1` 的容积, 则用一个变量接收这个方法返回的值即可得到, 其用法如下。

```
double box1Volume=box1.BoxVolume();
```

### 4. 类的构造函数

构造函数一般用来在对象初始化时对对象中的数据赋初始化值, 但这些代码均要写在构造函数中, 且构造函数与类的名称相同。下面在 `Box` 类的定义中加入构造函数。

```
class Box { //定义一个类
    public double length;
    public double width;
    public double height;
    Box(){
        length=10;
        width=10;
        height=10;
    }
    public double boxVolume(){
        double volumeTemp;
```

```

        volumeTemp=length*height*width;
        return volumeTemp;
    }
}

```

这样, 当用 `new Box()` 创建一个新的 `Box` 类实例时, 就会自动赋予这个对象的 `length`、`width`、`height` 三个数据初值为 10。

## 5. 同名的方法

在 `Java` 中, 如果方法的参数不同, 即便是方法名相同, 也代表不同的方法。类中的方法, 包括构造函数和其他方法都可以同名, 如果在 `Box` 类定义中加入不同的构造函数, 会怎么样呢? 参见下例。

```

class Box { //定义一个类
    public double length;
    public double width;
    public double height;
    Box(){
        length=10;
        width=10;
        height=10;
    }
    Box(double l,double w,double h){
        length=l;
        width=w;
        height=h;
    }
    public double BoxVolume(){
        double volumeTemp;
        volumeTemp=length*height*width;
        return volumeTemp;
    }
}

```

上述 `Box` 类的定义中定义了两个不同的构造函数, 虽名称相同, 但参数不同。再来看如下的对象声明代码。

```

Box box1=new Box();
Box box2=new Box(20,20,20);

```

代码中声明了两个 `Box` 类的对象实例 `box1` 和 `box2`, 两者均通过构造函数对数据进行了初始化, 其中, `box1` 的 `length`、`width`、`height` 三个数据初值均为 10; `box2` 的 `length`、`width`、`height` 三个数据初值均为 20。

## 6. this

有时一个方法需要引用调用它的对象, 这时 `this` 关键字就可以派上用场了。如 `Box` 类的构造函数可以改写为

```

public Box(double l,double w,double h){
    this.length=l;
    this.width=w;
    this.height=h;
}

```



## 7. 类的继承

类的继承是面向对象技术的重要特点之一，它可以大大提高程序代码的可复用性。在 Java 中，被继承的类叫超类（superclass），继承超类的类叫做子类（subclass），它继承了超类的数据和方法（并不是所有的数据和方法都会继承，为 public 和 protect 的才会继承，private 的不会继承，但如果都处在同一个包中则默认情况下都会继承），并可以在其中加入自己特有的数据和方法。

声明一个子类使用 **extends** 关键字，声明的语法格式如下。

```
class 子类类名 extends 超类类名{
    //子类程序代码
}
```



**注意** Java 中不支持多个超类的继承，这与 C++ 不同。

下面来看一个类的继承应用的例子。

```
class SuperClass_A{//超类
    public int i;
    public int j;
    public int add_i_j(){
        return i+j;
    }
}
class SubClass_A extends SuperClass_A{//子类
    public int k;
    public int add_i_j_k(){
        return i+j+k;
    }
}
```

在程序中首先声明了一个超类 **SuperClass\_A**，其中包括两个数据和一个把这两个数据相加的方法 **add\_i\_j()**；类 **SubClass\_A** 继承了超类 **SuperClass\_A**，所以在子类 **SubClass\_A** 中拥有超类 **SuperClass\_A** 声明的所有数据和方法，因此程序中 **SubClass\_A** 调用了 **i**、**j**，而这些数据是在超类中声明的。

**super** 可以用来访问超类的构造方法和被子类所隐藏的方法，如果子类中有方法与超类中的方法名称和参数相同，则超类中的方法就被隐藏起来，也就是说在子类中重载了父类中的方法。引用父类中所隐藏的语法格式如下：

```
super(参数列表)或 super.方法名(参数列表)
```

下面通过一段程序代码来理解 **super** 的用法。

```
class SuperClass_A{//超类
    public int i;
    public int j;
    public SuperClass_A(){
        j=1;
    }
}
```

```

    }
}
class SubClass_A extends SuperClass_A{ //子类
    public int i;
    SubClass_A (int a,int b){
        super();
        i=b;
    }
}

```

如果在声明 SubClass\_A 对象时用构造函数 SubClass\_A(2,3), 则首选通过 super()把 j 的值设为 1, 再把子类的 i 值设为 3。

## 8. 抽象类

抽象类定义了类的结构, 但并不定义完整的实现方法, 它要求继承它的子类必须实现方法。声明一个抽象类的语法格式如下。

```

abstract class 类名 {
    //类中的程序代码
}

```



**注意** 一个抽象类不能通过 new 直接实例化。

下面是一个抽象类的例子。

```

abstract class SuperClass_A{//抽象类
    public int i;
    public int j;
    abstract public int add_i_j();
}
class SubClass_A extends SuperClass_A{ //子类
    public int add_i_j(){
        return i+j;
    }
}

```

抽象类 SuperClass\_A 中定义了方法 add\_i\_j(), 却没有定义如何实现这个方法, 此方法在子类 SubClass\_A 中得以实现。

## 9. Object 类

在 Java 中有一个特殊的类: Object 类, 它是所有其他类的超类, 也就是说, 其他的类都是它的子类, 因此通过 Object 类的实例可以引用任何一个其他类型的对象。

### 4.2.6 String 类

Java 的字符串常量用 " " 括起来, 是一串字符串, 这一点与 C/C++ 类似。但 Java 中的字符是作为一个 String 对象来处理的, 而非数组。

在 JSP 程序设计中, 大量的信息是通过字符串来进行交互的, 为此一定要掌握好 String 类



的使用。Java 为 String 类提供了丰富的功能特性。

### 1. 构造函数

String 类支持几种构造函数，主要用来初始化 String 对象的值。可以用空的构造函数，表示创建一个新的 String 实例，它是一个空字符串，如下所示。

```
String s=new String();
```

也可以用一个字符型数组来构造一个字符串，如下所示的一个例子，就是用字符数组 charTemp 作为字符串 s 的初始值。

```
char charTemp[]={'a','b','c','d'};  
String s=new String(charTemp);
```

也可以用下面的构造函数把一个字符数组的一部分作为初始值，格式如下。

```
String(char chars[],int startIndex,int numChars);
```

第一个参数为字符数组，第二个参数为指定开始处的数组下标，第三个参数为从第二个参数开始处所取字符的个数。



**注意** 数据的下标从 0 开始。

如下所示的程序代码，用字符 bc 来初始化 s。

```
char charTemp[]={'a','b','c','d'};  
String s=new String(charTemp,1,2);
```

还可以用一个字符串来初始化另一个字符串，格式如下。

```
String(String stringObject)
```

### 2. 求字符串长度

求字符串长度用 String 类的方法 length()，如下的语句可求出字符串 s 的长度。

```
int slength=s.length();
```

length()方法返回的是整型数值。

### 3. 字符串连接

字符串连接使用“+”即可，将返回连接后的字符串，当和其他数据类型连接时，返回的数据类型也为字符串。代码如下所示。

```
out.println("输出一个布尔值："+true);
```

此时在浏览器中输出一个字符串“输出一个布尔值：true”。

### 4. 字符串转换

差不多每一个类都有 toString()方法，但这往往不够，大多数情况下需要重载这一方法。对于包装类可以用 toString()方法将其转换成对应的 String 类。对于整型数据还可以使用 toBinaryString(int i)、toHexString(int i)、toOctalString(int i)分别转换为二进制、十六进制和八进制

字符串的形式。

用 `String` 类的 `toCharArray()` 方法可以将字符串转换为一个字符数组，语法格式如下。

```
char[] toCharArray()
```

## 5. 字符截取

通过 `charAt()` 方法，可以得到字符串中指定位置的字符，其格式如下。

```
char charAt(int where)
```



**注意** 字符串的下标从 0 开始。

参数 `where` 是获得的字符在字符串中的位置。如下列程序代码。

```
char charTemp;  
charTemp="abcdef".charAt(2);
```

`charTemp` 中得到的字符为 `c`，因为字符串的下标从 0 开始。

通过 `getChars()` 方法可以一次获得字符串中的多个字符。方法的语法格式如下。

```
void getChars(int sourceStart,int sourceEnd,int target[],int targetStart)
```

第一个参数 `sourceStart` 指出了字符串截取所开始的位置；第二个参数 `sourceEnd` 指出了字符串截取所结束的位置；第三个参数指出目标（即接收）字符数组；第四个参数指出目标字符数据接收的开始下标。

### 【实例 4-5】字符截取程序实例

#### StringgetChars.jsp

```
<%@ page contentType="text/html;charset=gb2312"%>  
<%@ page import="java.util.*"%>  
<html>  
<body>  
<%  
    String s="this is a student.";  
    int startPoistion=1;  
    int endPoistion=7;  
    char pointChars[]=new char[endPoistion-startPoistion];  
    s.getChars(startPoistion,endPoistion,pointChars,0);  
    out.println(pointChars);  
%>  
</body>  
</html>
```

程序在浏览器中的输出结果是 “his is”。

另外，`getBytes()` 方法可把字符串中的字符存放于字节数组中，语法格式如下。

```
byte[] getBytes()
```

在将字符串输出到一个不支持 `Unicode` 的环境时，`getBytes()` 方法极为有用，现今大多数基于 `Internet` 的文本交换以 `ASCII` 编码为主，`getBytes()` 就可以派上用场，如做中文处理，在本章后面的小节中还将专门介绍在 `Java` 中如何处理中文字符。



## 6. 字符串的比较

`equals()`方法用来对字符串进行比较，如果要忽略大小写的差异，则可用 `equalsIgnoreCase()` 方法，两者的语法格式如下。

```
boolean equals(Object str)
boolean equalsIgnoreCase(String str)
```

本方法由字符串调用，如果两者的字符与长度都相同，则返回 `true`，否则返回 `false`。



**注意** `equals()`与“`==`”不同，`equals()`方法比较的是两个 `String` 对象中的字符，“`==`”比较的是看两者是否引用相同的实例。

`regionMatches()`方法将一个字符串中的一部分与另一个字符串中的一部分进行比较，它的重载形式允许忽略大小写。两者的语法格式如下。

```
boolean regionMatches(int startIndex, String str2, int str2startIndex, int numChars)
boolean regionMatches(boolean ignoreCase, int startIndex, String str2, int str2startIndex, int numChars)
```

两者的参数说明：`startIndex` 参数指出调用字符串开始比较的下标，比较的字符串由 `str2` 参数指出；开始比较的下标由 `str2startIndex` 参数指定，`numChars` 参数是比较的字符个数；第二种调用方法中的 `ignoreCase` 参数如果设为 `true` 则表示忽略大小写，如果设为 `false` 则表示区分大小写。

`startsWith()`和 `endsWith()`方法用于判断字符串是否以指定的字符串开始或结束。两者的语法格式如下：

```
boolean startsWith(String str)或 boolean startsWith(String str, int startIndex)
boolean endsWith(String str)
```

如果判断为“是”则返回 `true`，“否”则返回 `false`。上面所示 `startsWith()`方法的 `startIndex` 参数指出了开始比较的下标。如下列程序代码。

```
String s="student"
boolean bstartwith;
bstartwith=s.startsWith("tud",1);
```

`bstartwith` 的值最终为 `true`，因为 `s` 从下标 1 开始与字符串“`tud`”相同。

`compareTo()`方法用来比较两个字符串，不仅可以比较是否相等，而且还可以根据编码字典来比较字符串的大小，从第一个字符开始比较，如果第一个相同再接着比较第二个字符。`compareTo()`方法的语法格式如下。

```
int compareTo(String str)
```

当返回值小于 0 时，表示调用字符串小于 `str`；当返回值等于 0，表示两者相同；当返回值大于 0，表示调用字符串大于 `str`。`compareTo()`方法常常被用于各种排序算法中。

## 7. 查找字符串

要在字符串中查找指定的字符或子字符串时，可以用 `String` 类的两个方法，其方法如下。

`indexOf()`——查找字符或子字符串在字符串中首次出现的位置。

`lastIndexOf()`——查找字符或子字符串在字符串中最后一次出现的位置。

如果查找成功,方法返回字符或子字符串在字符串中出现位置的下标,如果失败则返回-1。这两个方法有多种用法,其语法格式分别表述如下。

```
int indexOf(char ch)
int lastIndexOf(char ch)
int indexOf(String str)
int lastIndexOf(String str)
int indexOf(char ch,int startInex)
int lastIndexOf(char ch,int startInex)
int indexOf(String str,int startInex)
int lastIndexOf(String str,int startInex)
```

上述参数中, `ch` 表示要查找的字符, `str` 表示要查找的子字符串, `startIndex` 是从字符串中开始查找的下标。对于 `indexOf()` 方法,从 `startIndex` 下标处开始查找,直到找到字符串结束。对于 `lastIndexOf()` 方法,则从 `startIndex` 下标处开始查找,到字符串开始处即下标为 0 处。

下面来看一个字符查找操作的程序实例。

#### 【实例 4-6】查找字符串程序实例

findString.jsp

```
<%@ page contentType="text/html; charset=gb2312"%>
<html>
<body>
<%//查找字符串程序示例 Java 程序片
String s=new String();
s="I am a student,he is also a student.";
out.println("字符串 s 为: "+s+"<br>");
out.println("在 s 中第一个 a 出现的下标是: "+s.indexOf('a')+"<br>");
out.println("在 s 中最后一个 a 出现的下标是: "+s.lastIndexOf('a')+"<br>");
out.println("在 s 中第一个 en 出现的下标是: "+s.indexOf("en")+"<br>");
out.println("在 s 中最后一个 en 出现的下标是: "+s.lastIndexOf("en")+"<br>");
out.println("在 s 中从下标 4 开始第一个 a 出现的下标是: "+s.indexOf('a',4)+"<br>");
out.println("在 s 中从下标 4 开始最后一个 a 出现的下标是: "+s.lastIndexOf('a',4)+"<br>");
out.println("在 s 中从下标 14 开始第一个 en 出现的下标是: "+s.indexOf("en",14)+"<br>");
out.println("在 s 中从下标 14 开始最后一个 en 出现的下标是: "+s.lastIndexOf("en",14)+"<br>");
%>
</body>
</html>
```

从程序代码中可以看到, `s` 中第一个 `a` 出现的下标为 2, 如果用 `lastIndexOf()` 方法,则从后往前看,第一个 `a` 出现的下标是 26, 查找子字符串依此类推;方法中有了 `startIndex` 参数后,就从指定的这个参数的下标位置开始查找,所以从下标 4 开始第一个 `a` 出现的下标是 5,而对于 `lastIndexOf()` 方法,则从此参数指定的下标处开始往前查找,故得到的下标为 2, 查找子字符



串依此类推。

实例 4-6 查找字符串程序示例的运行结果如图 4-5 所示。

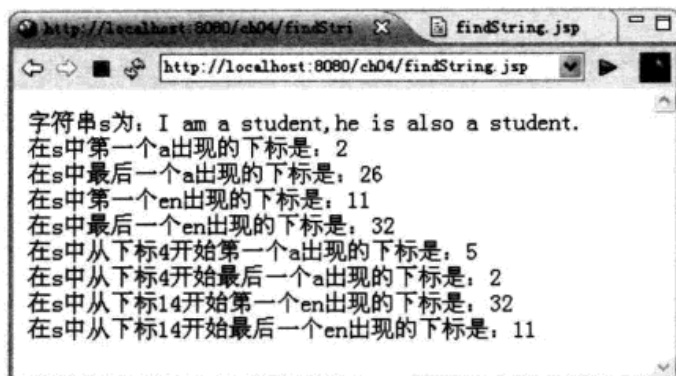


图 4-5 实例 4-6 查找字符串程序的运行结果



**注意** 字符串的下标从 0 开始。

## 8. 修改字符串

用来修改字符串的方法有多个，包括截取子字符串的方法 `substring()`、连接两个字符串的方法 `concat()`、替换字符方法 `replace()` 及去除空格方法 `trim()`。

用 `substring()` 方法可以截取子字符串，使用的语法格式有两种。

```
String substring(int startIndex)
String substring(int startIndex,int endIndex)
```

上述两种方法中，参数 `startIndex` 指定了子字符串开始的下标，参数 `endIndex` 指定了子字符串结束的下标。第一种形式将返回从下标 `startIndex` 开始直至末尾的子字符串，第二种形式将返回从下标 `startIndex` 开始到下标 `endIndex` 结束的子字符串。

`concat()` 方法可以用来连接两个字符串，并创建一个新的 `String` 类的对象，它是连接后的字符串，其用法如下。

```
String concat(String str)
```

`replace()` 方法可以用一个字符代替在字符串中出现的所有这个字符，其用法如下。

```
String replace(char originalChar,char replaceChar)
```

该方法用字符 `replaceChar` 代替字符 `originalChar`，返回替换后的字符串。

`trim()` 方法用来去除字符串中的空格，这个方法在接收字符数据时特别有效，因为用户输入数据时往往会不经意地输入空格。

```
String trim()
```

该方法返回去除字符串中空格后的字符串。

## 9. 改变大小写

改变字符串的大小写用方法 `toLowerCase()` 和 `toUpperCase()`。`toLowerCase()` 方法把字符串转换为小写，`toUpperCase()` 将字符串转换为大写。两者调用的语法格式如下。

```
String toLowerCase()
String toUpperCase()
```

### 4.2.7 StringBuffer 类

`String` 类表示的字符串是定长的，而 `StringBuffer` 类提供了可变长的字符串，同时还提供了大量的字符串功能。在 `StringBuffer` 类的字符串中可以再插入字符，此时，`StringBuffer` 会自动增加内存空间，这在 `String` 中是无法做到的。

#### 1. 构造函数

`StringBuffer` 的构造函数有以下三种形式。

```
StringBuffer()
StringBuffer(int buffersize)
StringBuffer(String str)
```

第一种形式，采用默认的构造函数，会给字符串预留 16 个字符的空间；第二种形式中的 `buffersize` 用来设置缓冲区的大小；第三种形式中的数 `str` 用来初始化 `StringBuffer` 中的内容。

#### 2. 长度运算

`length()` 方法用来得到字符串的长度，`capacity()` 用来得到分配给字符串的容量，注意两者是不一样的，它通常比实际字符要预留更多的空间，以允许增加字符。两者调用的语法格式如下。

```
int length()
int capacity()
```

如果初始化了字符串后，要更改缓冲区大小可用 `ensureCapacity()` 方法，这样可以为新增字符串预先分配空间，因为再分配空间的代价是比较大的，而且频繁地分配空间会产生碎片。此方法的语法格式如下。

```
void ensureCapacity(int capacitysize)
```

其中，`capacitysize` 参数用来指定缓冲区的大小。

要改变字符串的长度可用 `setLength()` 方法，语法格式如下。

```
void setLength(int length)
```

其中，参数 `length` 指出了要设置的字符串长度，如果 `length` 长度大于原长度，则在新加入长度的空间中设为空格，如果 `length` 长度小于原长度，则字符串后面小于原长度部分的子字符串将会丢失，使用此方法时要特别注意这一点。

#### 3. 获取与修改字符

使用 `charAt()` 方法可从 `StringBuffer` 类的字符串中获取指定的字符；通过 `setCharAt()` 方法可以修改 `StringBuffer` 类字符串中指定的字符。两者的用法如下。

```
char charAt(int position)
void setCharAt(int position, char ch)
```

第一个方法 `charAt()` 中的参数 `position` 指出了要获取的字符在字符串中的下标；第二个方法 `setCharAt()` 中第一个参数 `position` 是要更改的字符的下标，第二个参数 `ch` 是更改后的字符。



#### 4. 获取子字符串

用 `getChars()` 方法把 `StringBuffer` 中的子字符串复制给一个数组，使用的形式如下。

```
void getChars(int sourceStart,int sourceEnd,char target[],int targetStart)
```

这个方法的第一个参数 `sourceStart` 是 `StringBuffer` 中要提取的字符串开始的下标；第二个参数 `sourceEnd` 是结束的下标；第三个参数 `target[]` 是接收子字符串的字符数组；第四个参数 `targetStart` 是开始接收数组的下标。



**注意** 使用此方法前一定要确保 `target[]` 数组的长度足够长，以容纳子字符串的内容。

另外，`substring()` 方法也可用于获取 `StringBuffer` 的一部分，它有如下两种调用形式。

```
String substring(int startPosition)
```

```
String substring(int startPosition, int endPosition)
```

前一种形式用于返回从 `startPosition` 下标开始到末尾的子字符串；第二种形式用于返回从下标 `startPosition` 开始到下标 `endPosition` 结束的子字符串。

#### 5. 追加字符或字符串

在 `StringBuffer` 尾部追加字符或字符串可使用方法 `append()`，调用的方法有多种，统一表述如下。

```
StringBuffer append(Object obj)
```

其中，参数 `obj` 可以是任意类型，比较常用的是一些简单数据类型以及 `String` 类，该方法返回追加后的 `StringBuffer`。

#### 6. 插入字符

用 `insert()` 方法可将一个字符串插入到另一个字符串中。使用的方法有许多种，常用的方法形式如下。

```
StringBuffer insert(int insertIndex,char ch)
```

```
StringBuffer insert(int insertIndex,String str)
```

第一种形式用来在 `insertIndex` 处插入一个字符 `ch`；第二种形式用来在 `insertIndex` 处插入一个字符串 `str`。

#### 7. 字符串翻转

要实现 `StringBuffer` 字符串的翻转可用方法 `reverse()`，其调用的语法格式如下。

```
StringBuffer reverse()
```

这个方法返回翻转后的字符串。

#### 8. 删除字符和字符串

删除字符和字符串的方法有两种，它们调用的语法格式如下。

```
StringBuffer delete(int startPosition,int endPosition)
StringBuffer deleteCharAt(int deletePostion)
```

第一个方法 `delete()` 用于删除从 `startPosition` 下标开始到 `endPositon` 下标结束的字符串，返回删除后的 `StringBuffer` 对象；第二个方法 `deleteCharAt()` 用于删除位于 `deletePosition` 下标处的字符，这两个方法均返回删除后的 `StringBuffer` 对象。

## 9. 替换字符串

方法 `replace()` 完成用一个字符串取代 `StringBuffer` 中的部分字符串的功能。其调用方法如下。

```
StringBuffer replace(int startPosition,int endPosition,String str)
```

该方法把 `StringBuffer` 中从下标 `startPosition` 处开始到下标 `endPosition` 处结束的字符串替换为 `str`，返回替换后的 `StringBuffer` 对象。

### 【实例 4-7】StringBuffer 综合应用程序实例

#### StringBufferApp.jsp

```
<%@ page contentType="text/html;charset=gb2312"%>
<html>
<body>
<!--StringBuffer 综合应用示例 Java 程序片
StringBuffer s=new StringBuffer("He is also a student.");
out.println("StringBuffer 字符串 s 为: "+s+"<br>");
out.println("s 的长度为: "+s.length()+"<br>");
out.println("s 的容量为: "+s.capacity()+"<br>");
out.println("s 的第 3 个字符为: "+s.charAt(2)+"<br>");
out.println("s 的第 3 至第 6 个字符的子串为: "+s.substring(2,5)+"<br>");
out.println("s 末尾追加串后为: "+s.append("Me,too.")+"<br>");
out.println("s 翻转后的串为: "+s.reverse()+"<br>");
%>
</body>
</html>
```

该程序首先用构造函数初始化了 `StringBuffer` 对象 `s`，然后显示这个字符串，求出其容量和长度，并从中得到第 3 个字符，注意下标是从 0 开始的，所以参数为 2，运用了获取子字符串方法，追加了一个字符串，最后进行了翻转处理。本示例仅仅起到抛砖引玉的作用，理解了这些方法的应用，相信读者应用其他的方法并不难。

程序的运行结果如图 4-6 所示。

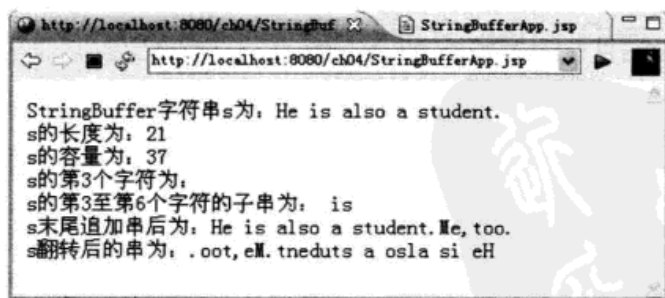


图 4-6 实例 4-7 StringBuffer 综合应用程序的运行结果



`String` 类和 `StringBuffer` 类是 JSP 程序设计中经常要用到的，希望读者多做练习，以融会贯通，在本章后面，配备了相关的练习题，读者可自行编写代码。

### 【实例 4-8】省略显示长字符串实例

在许多网站的页面上都可以看到省略长字符串的情况。如在网站的首页上显示出一些文章的标题，然而这些文章的标题有长有短，如果 HTML 表格固定列宽则文章的标题较长时就会自动换行显示，如果没有固定列宽就会把显示内容变得杂乱无章，甚至使页面变形。通常情况下的做法是，截取标题字符串的前面一部分，然后在后面加“...”。看如下实例程序源代码。

strTruncate.jsp

```
<%@ page contentType="text/html;charset=gb2312"%><!--JSP 指令标签-->
<%@ page import="java.util.*"%> <!--JSP 指令标签-->
<html><!--HTML 标记符-->
<head><title>长字符串截取示例</title></head>
<body>
<%! //截取字符串函数,返回处理后的字符串
//参数说明:source 表示需要截断的字符串
//len 表示要截取的字节数
//delim 表示截取后附加在后的字符串
public static String strTruncate(String source, int len, String delim) {
    if (source == null) return null;//字符串为空不做处理
    int start, stop, byteLen;
    int alen = source.getBytes().length;//得到需要截断的字符串的字节数
    if (len > 0) {
        if (alen <= len) return source;//如果比要截取的字节数还小,不做处理
        start = stop = byteLen = 0;
        while (byteLen <= len){
            if (source.substring(stop,stop+1).getBytes().length == 1){
                //单字节字符处理
                byteLen += 1;
            }else{//双字节字符处理
                byteLen += 2;
            }
            stop++;
        }
        StringBuffer sb = new StringBuffer(source.substring(start, stop-1));
        if (alen > len) sb.append(delim);//加入附加在后的字符串
        return sb.toString();
    }
    return source;
}
%>
<%
String s1=new String("[2011年3月1日]物联网应用技术专业正式获得教育部批准!");
String s2=new String("[2011年6月1日]我系一本新的著作正式在电子工业出版社出版!");
String s3=new String("[2011年4月30日]SOA项目组形成新一代集成平台版本。");
out.println("长字符串截取示例: <br>");
out.println(strTruncate(s1,40,"...")+"<br>");
out.println(strTruncate(s2,40,"...")+"<br>");
out.println(strTruncate(s3,40,"...")+"<br>");
%>
```

```
%>
</body><!--HTML 标记符-->
</html>
```

其中，`strTruncate` 用来截取字符串，并用指定的字符串附加到处理完后的字符串的末尾，本例声明了三个较长的字符串，然后调用函数做截取处理，并将其显示出来。运行的结果如图 4-7 所示。

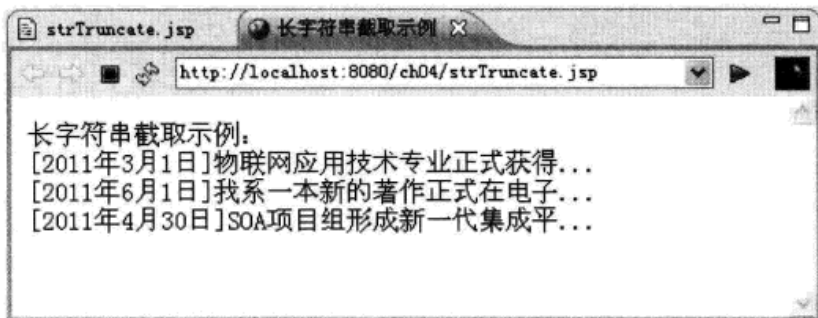


图 4-7 实例 4-8 省略显示长字符串运行结果

实际应用时，长字符串文本常从数据库中取出，并在处理后的字符串上加上超链接，当用户单击时，则打开一个窗口显示此标题指向的文章内容。

## 4.3 运算符与表达式

Java 中的运算符与 C/C++ 中基本相同，但有一些小的差异，使用的时候要特别注意。Java 中的运算符特别丰富，分为四类：算术运算符、关系运算符、布尔运算符和移位运算符。

### 4.3.1 算术运算与表达式

在数学表达式中使用算术运算符。Java 中的算术运算符包括如下一些：+（加法）、-（减法或一元负号）、\*（乘法）、/（除法）、%（取模运算）、++（递增运算）、--（递减运算）、+=（加后赋值运算）、-=（减后赋值运算）、\*=（乘后赋值运算）、/=（除后赋值运算）和%=（取模后赋值运算）。

算术运算专用于数学运算，不能用于布尔运算，但 `char` 类型也可用算术运算，但提倡读者在实践工程中不要这么做，以免带来不必要的麻烦。

取模运算%的结果是进行整数除法后的余数，也能用于浮点型（这一点与 C/C++ 不同）。

++、--、+=、-=、\*=、/=、%= 这些运算实际上是在运算之后再进行一个赋值操作，如：

`a++` 或 `++a`，这个表达式等价于 `a=a+1`；

`a--` 或 `--a`，这个表达式等价于 `a=a-1`；

`a+=b`，这个表达式等价于 `a=a+b`；

`a-=b`，这个表达式等价于 `a=a-b`；

`a*=b`，这个表达式等价于 `a=a*b`；



$a/=b$ , 这个表达式等价于  $a=a/b$ ;

$a\%=b$ , 这个表达式等价于  $a=a\%b$ 。

在 Java 的程序中经常会看到使用了这些符号的程序代码。使用++、—运算符时还要特别注意以下两种情况:

```
a=++b;
a=b++;
```

这两个表达式的结果是不一样的。因为  $a=++b$  实质上等价于如下的两句:

```
b=b+1;
a=b;
```

而  $a=b++$  实质上等价于:

```
a=b;
b=b+1;
```

两句语句执行的顺序不一样, 结果也会不一样。如果在这两句语句之前使  $a=10$ ,  $b=10$ , 则  $a=++b$  表达式运算完毕后, 得到  $a=11$ ,  $b=11$ ; 但  $a=b++$  表达式运算完毕后, 会得到  $a=10$ ,  $b=11$ 。请读者注意两者在应用时的这一点细小的差别。

#### 【实例 4-9】算术表达式综合运用实例

mathApp1.jsp

```
<%@ page contentType="text/html; charset=gb2312"%>
<html>
<body>
<%// 算术表达式综合运用实例 Java 程序片
    int a=10,b=3,c;
    out.println("a="+a+",b="+b+"<br>");
    out.println("a/b="+ (a/b) + "<br>");
    out.println("a%b="+ (a%b) + "<br>");
    a++;
    out.println("a++后 a="+a+"<br>");
    c=b++;
    out.println("c=b++后 c="+c+"<br>");
    b--; //恢复 b 的值
    c=++b;
    out.println("c=++b 后 c="+c+"<br>");
%>
</body>
</html>
```

程序中, 首先声明了三个整型变量  $a$ 、 $b$ 、 $c$ , 并给  $a$  和  $b$  赋予初始值。然后进行了除法运算、取模运算和增 1 运算, 特别演示了  $c=b++$  与  $c=++b$  表达式的区别。从图 4-8 的算术表达式综合运用实例程序的运行结果可以看出两者得到  $c$  的值不同, 这主要取决于是否先进行增 1 运算还是后进行增 1 运算。



图 4-8 实例 4-9 算术表达式综合运用实例程序运行结果

### 4.3.2 关系运算与表达式

关系运算主要是对值与值之间进行比较后得到的关系，其结果为布尔值，在程序中做条件判断时比较常见。

关系运算符包括：`=`（等于）、`!=`（不等于）、`>`（大于）、`<`（小于）、`>=`（大于或等于）、`<=`（小于或等于）。简单数据类型都可以进行关系运算。

**注意** 在 C/C++ 的判断中，把非 0 值作为 `true`，把 0 值作为 `false`，而在 Java 中则不同，判断只接受 `true` 或 `false` 这两种布尔值。

### 4.3.3 布尔运算与表达式

参与布尔运算的运算数只能是布尔型数据，结果也是布尔型数据。

布尔运算符有：`&`（逻辑与）、`|`（逻辑或）、`^`（异或）、`||`（短路或）、`&&`（短路与）、`!`（逻辑反）、`&=`（逻辑与后赋值）、`|=`（逻辑或后赋值）、`^=`（逻辑异或后赋值）、`==`（等于）、`!=`（不等于）、`?:`（三元运算符）。

`&`、`|`、`^`、`!` 运算的真值表如表 4-2 所示。

表 4-2 布尔运算真值表

A	B	A&B	A B	A^B	!A
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

从表 4-2 中可以看出，在 `||` 运算中，如果第一个运算数的值为 `true`，则第二个运算数将不再计算，其结果为 `true`；在 `&&` 运算中，如果第一个运算数的值为 `false`，则第二个运算数将不再计算，其结果为 `false`。

三元运算符（`?:`）类似于 `if-then-else` 的句型，它的用法如下。

```
expression1?expression2:expression3
```

其中，`expression1` 必须是一个逻辑表达式，它的含义是，当 `expression1` 的值为 `true` 时求



表达式 `expression2` 的值，否则求表达式 `expression3` 的值。

如果 `a=10`, `b=12`, 来看下面的语句。

```
a>b?c=3:c=4;
```

运算结束后 `c` 的值将是 4。

### 4.3.4 位运算与表达式

Java 中的位运算主要针对整型数据。位运算符包括：`~`（按位非）、`&`（按位与）、`|`（按位或）、`^`（按位异或）、`>>`（右移）、`>>>`（右移，左边空出的位以 0 补充）、`<<`（左移）、`&=`（按位与后再赋值）、`|=`（按位或后再赋值）、`^=`（按位异或后再赋值）、`>>=`（右移后再赋值）、`>>>=`（右移后再赋值，左边空出的位以 0 补充）、`<<=`（左移后再赋值）。不过，位运算在 JSP 应用程序设计里很少用到。



**注意** Java 中使用补码来表示二进制数。“`~`”运算符是一元运算符。

## 4.4 程序控制逻辑

控制逻辑用来控制程序执行的顺序与流程。在 Java 中控制逻辑分为选择分支、循环和跳转三种。Java 中的控制语句与 C/C++ 中的类似，只是在 `break` 语句与 `continue` 语句上稍有差异。

### 4.4.1 选择分支

#### 1. if 语句

if 语句的程序控制逻辑如图 4-9 所示。

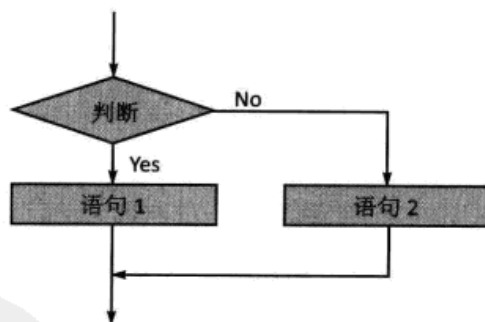


图 4-9 if 语句的控制逻辑

程序执行的路径一分为二，判断表达式为真是一条，为假是另一条，它的语法格式如下。

```
if(判断表达式)
    statement1;
else
    statement2;
```

其中, `statement1` 和 `statement2` 可以是单个语句, 也可以是语句块, 两者永远不可能同时被执行。如果是语句块则要用花括号 “{}” 括起来。

`if` 语句可以嵌套使用, 嵌套使用时要注意花括号的匹配问题, 它们总是配对使用的。分支较多时, 建议使用 `switch` 语句, 这样将使程序更具可读性。此外, 将程序语句缩格编写, 是一种良好的编程习惯, 这样将提高程序的可读性。

## 2. switch 语句

如果分支较多, 就可以采用 `switch` 语句, 其控制逻辑如图 4-10 所示。

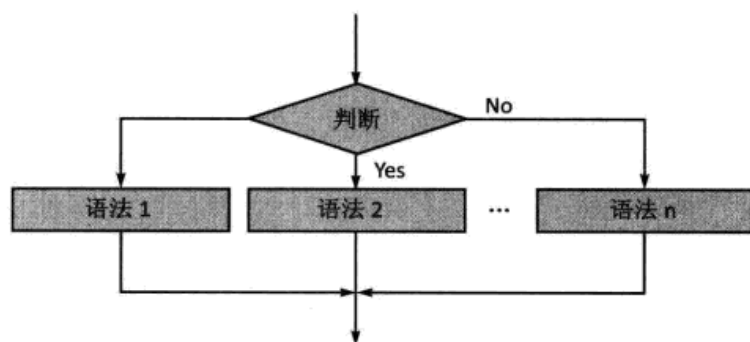


图 4-10 `switch` 语句的控制逻辑

其应用的语法格式如下:

```

switch(表达式)
{
    case value1:
        statement1;
        break;
    case value2:
        statement2;
        break;
    ...
    case valuen:
        statementn;
        break;
    default:
        default_statement;
}
  
```

这里, 表达式必须为 `byte`、`short`、`int` 或 `char` 数据类型, 每个 `case` 语句后的 `value` 必须是一个与表达式类型兼容的特定常量。语句的执行过程如下, 用表达式的值与 `case` 后的 `value` 值进行比较, 如果找到一个与之匹配的, 则执行相应的语句后退出 `switch` 语句, 如果没找到, 则执行 `default` 后的语句再退出 `switch` 语句。



**注意** `break` 语句是可选的, 如果没有 `break` 语句, 则执行完 `statement` 语句后将继续执行后面的 `statement` 语句。



## 【实例 4-10】switch 语句示例

SwitchApp1.jsp

```
<%@ page contentType="text/html;charset=gb2312"%>
<html>
<body>
<form name="form1" action="SwitchApp1.jsp" method="post">
请选择一种颜色:
<select name="ys">
  <option value="red">红色</option>
  <option value="green">绿色</option>
  <option value="blue">蓝色</option>
</select>
<input type="submit" name="submit" value="提交">
</form>
<%//switch 语句应用示例 Java 程序片
String s=null;
s=request.getParameter("ys");
if(s!=null){
    switch(s.charAt(0)){
        case 'r':
            out.println("你选择的是红色。");
            break;
        case 'g':
            out.println("你选择的是绿色。");
            break;
        case 'b':
            out.println("你选择的是蓝色。");
            break;
        default:
            out.println("你没有选择。");
    }
}else
    out.println("你没有选择。");
%>
</body>
</html>
```

该程序用一个包含下拉列表的表单提交选择的颜色,再根据提交的颜色调用 switch 语句分别输出选择的是什么颜色,判断是什么颜色是根据颜色的首字母来判断的,这里只有三种颜色。程序的运行结果如图 4-11 所示。



图 4-11 实例 4-10 switch 语句示例程序的运行结果

## 4.4.2 循环

循环语句有 `while`、`do-while` 和 `for` 三种，都是常用的。

### 1. while

`while` 语句的控制逻辑如图 4-12 所示。

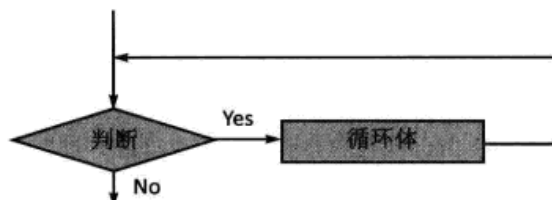


图 4-12 `while` 语句的控制逻辑

`while` 语句的用法如下：

```
while(condition){
    //循环体中的代码块
}
```



**注意** `condition` 必须是一个布尔表达式。

如果条件表达式 `condition` 为真，则继续执行循环体，如果为假，则退出循环。在 `while` 语句中，如果循环体只有一句语句，则花括号可以去掉。

条件表达式 `condition` 中的某些变量常常是循环体中的更改值，以利于程序找到循环的出口。没有出口的循环称为无限循环，程序设计时一定要避免这种情况。

### 2. do-while

`do-while` 语句的控制逻辑如图 4-13 所示。

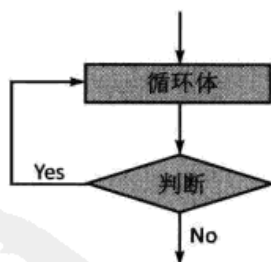


图 4-13 `do-while` 语句的控制逻辑

`do-while` 语句的用法如下：

```
do{
    //循环体中的代码块
}while(condition);
```



**注意** `condition` 必须是一个布尔表达式。



这样，不论 `condition` 表达式的值是真还是假，循环体至少会执行一次，并且先执行循环体再判断表达式，这一点正是与 `while` 语句的最大区别。

### 3. for

`for` 循环是一种功能强大而使用特别灵活的循环控制语句。使用 `for` 循环的通用格式如下：

```
for(initialization;condition;iteration){
    //循环体中的代码块
}
```

如果循环体只有一句语句，则花括号可以略去。循环执行的过程如下：第一次执行时，先进行初始化，即执行 `initialization`，一般用来初始化 `condition` 中变量的值，检查 `condition` 表达式。如果结果为真，则执行循环体，如果为假，则退出循环。每次执行完循环体都要执行 `iteration` 语句，一般用来对 `condition` 中变量的值进行变更处理，如增量或减量。

在初始化语句 `initialization` 中，如果要给多个变量赋值，则赋值表达式之间用逗号“,”隔开。下面来看一个循环的综合应用实例。

#### 【实例 4-11】循环应用综合实例

xhApp1.jsp

```
<%@ page contentType="text/html;charset=gb2312"%>
<html>
<body>
<%//循环应用综合实例 Java 程序片
    out.println("求 1 至 100 的和。<br>");
    int i=1;
    int s=0;
    //用 while 循环语句求和
    while(i<=100){
        s=s+i;
        i++;
    }
    out.println("用 while 循环求得和为: "+s+"<br>");
    //数据重新初始化
    i=1;
    s=0;
    //用 do-while 循环语句求和
    do{
        s=s+i;
        i++;
    }while(i<=100);
    out.println("用 do-while 循环求得和为: "+s+"<br>");
    //数据重新初始化
    i=1;
    s=0;
    //用 for 循环语句求和
    for(;i<=100;i++) s+=i;
    out.println("用 for 循环求得和为: "+s+"<br>");
%>
</body>
</html>
```

该程序分别用三个循环实现从 1 至 100 的自然数求和。从上述代码中可以看出, while 和 do-while 循环在条件判断上并无多大差异, 只是 do-while 循环的循环体至少会执行一次, while 循环就不一定了, for 循环的程序代码特别简洁, 使用灵活。程序的运行结果如图 4-14 所示。

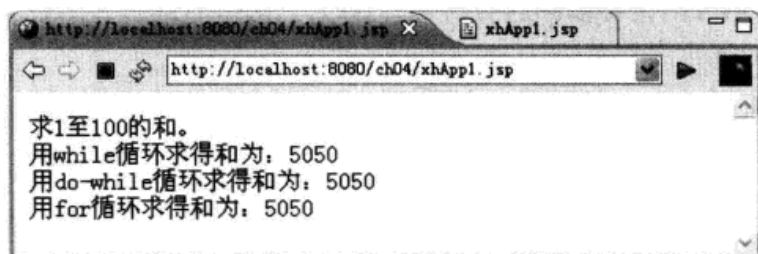


图 4-14 实例 4-11 循环应用综合实例程序运行结果

## 4.5 Java 程序片

在 JSP 中, 在 “<%” 和 “%>” 之间书写的程序代码称为 Java 程序片, 其实在前面已学的章节中已经多次用到。

一个 JSP 页面中可以有多个 Java 程序片。要注意的是, 在 Java 程序片中声明的变量在它们所在 JSP 页面的所用程序片及表达式中都有效。基于此, 可以把一个较大的程序片分成几个小的程序片, 还可在其中插入 HTML 语句, 以便编写的程序代码更具可读性。

在程序片中声明的变量只在页面有效, 是局部变量, 它在不同的客户访问同一个页面时, 此变量是不能共享的。但如果是在 “<%!” 和 “%>” 之间声明的变量就可以在不同的客户之间共享, 其有效范围是整个 Web 应用, 在服务器关闭后变量才会被释放。

用 “<%=” 和 “%>” 可以直接输出变量或表达式的值, 变量或表达式的值将作为一个字符串在浏览器中输出。这种方法在 JSP 编程中是比较常用的, 特别在与 HTML 标记混合编写时较为常用。来看下面的程序代码。

### 【实例 4-12】一个简单的计数器

simpleCounterApp1.jsp

```
<%@ page contentType="text/html; charset=gb2312"%>
<html>
<body>
<%!int counter=0;
    synchronized void counterFunction(){
        counter++;
    }
%>
<%counterFunction();%>
    网站计数器<br>
    你是第<%=counter%>位访问者
</body>
</html>
```



该程序首先在“<!”和“%>”之间声明了一个计数器变量和计数的方法，计数器变量将在客户之间共享，直至服务器关闭。计数方法实际上就是对计数变量做增 1 处理，在方法的前面加了 `synchronized` 关键字，这个关键字可防止客户在同时调用此方法更改计数变量的值时发生冲突，因此对方法作了串行化处理。一个简单的计数器程序的运行结果如图 4-15 所示。每刷新一次页面，数值会增 1，也可以同时打开多个浏览器窗口对此页进行测试，可发现变量是共享的。



图 4-15 实例 4-12 一个简单的计数器程序的运行结果

## 4.6 程序注释

适当地在 JSP 程序中加入注释可以增强程序的可读性，以方便维护人员进行维护。即便是对于程序编写人员自己，注释对调试程序和编写程序也可起到很好的帮助作用。程序员在程序中书写注释是一种良好的习惯，在平时学习和实践时要注意培养这种良好的习惯。

JSP 文件中的注释可分为两种：HTML 注释和 JSP 注释。

### 1. HTML 注释

在符号“<!--”与“-->”之间的是 HTML 注释，JSP 不会解释 HTML 注释，而直接把它交给客户端的浏览器，因此这种注释在浏览器中直接查看 JSP 源文件时，可以看到。格式如下：

```
<!-- 注释内容 -->
```

### 2. JSP 注释

JSP 注释的方法有以下四种，格式分别如下：

```
<%-- 注释内容 --%>
//注释内容
/*注释内容*/
/**注释内容*/
```

第一种注释会被 JSP 引擎忽略，一般用来对 Java 程序片作出说明；第二种注释是单行注释；第三种方式可以是单行注释，也可以是多行注释；第四种方式是 Java 所特有的 doc 注释。

注释在编写程序和调试程序中都是特别有用的，比如说在调试一段程序的其中几行程序时，可以把其他的程序进行注释，这样 JSP 引擎就不会编译它，这是一种比较常用的程序调试方法。

## 4.7 JSP 指令

### 4.7.1 page 指令

定义 JSP 页面的全局属性值时可使用 `page` 指令，一般把它放在页面的首部。如：

```
<%@ page contentType="text/html;charset=gb2312"%>
```

有了此句后，在 JSP 页面中就可以显示中文字符了。指令中的属性值可用单引号或双引号括起来，如果一个属性有多个值就用逗号隔开，在 `page` 指令中也只有 `import` 属性可以指定多个值，它用来导入一些程序中要用到的包或类，如：

```
<%@ page import="java.util.*","java.awt.*"%>
```

`page` 指令的语法格式如下所示：

```
<%@ page [language="脚本语言种类"] [import="包或类"] [contentType="MIME 类型"]
[session="true/false"] [buffer="缓冲区大小"]
[autoFlush="true/false"] [isThreadSafe="true/false"]
[info="text"] [errorPage="异常事件页面 URL"] [isErrorPage="true/false"] %>
```



**注意** `page` 指令中的 `contentType` 属性不能在同一个页面中被两次指定值。

`page` 指令的属性比较多，用方括号“[]”括起来的属性表示可选属性。下面来一一介绍它们。

#### 1. language

`language` 定义页面使用的脚本语言，默认情况下值为 `Java`，因此在编写 JSP 程序时，此属性不必设置。

#### 2. import

`import` 属性是常用的。它用来导入程序中要用到的包或类，此属性可以有多个值。无论是 `Java` 核心包中自带的类还是用户自行编写的类，都要在 `import` 中引用，这样才能在 JSP 程序中使用这个类。

默认情况下，JSP 文件中会自动导入如下类：

```
java.lang.*、javax.servlet.*、javax.servlet.jsp.*、javax.servlet.http.*
```

这里的“\*”表示该包下所有的类，自动导入的类就不必在 JSP 页面的 `page` 指令中再进行导入了。

#### 3. contentType

`contentType` 属性设置 JSP 页面的 `MIME`（Multipurpose Internet Mail Extension）类型。设置类型的格式为“`MIME 类型`”或“`MIME 类型; charset=编码`”。在 JSP 页面默认情况下设置的字符编码为 `ISO-8859-1`，即 `type="text/html; charset=ISO-8859-1"`。



#### 4. session

session 属性设置在 JSP 页面中是否需要使用 session 对象。如果为 false，则在 JSP 程序中不能使用 session 对象以及 scope=session 的 JavaBean 或 EJB。此属性的默认值为 true。

#### 5. buffer

buffer 的值可以有 none、8kb 或是给定的 kb 值，值为 none 表示没有缓存，直接输出至客户端的浏览器中，此属性用来设定 out 对象缓存处理的缓冲区的大小。

#### 6. autoFlash

autoFlash 属性设置当缓冲区已满时，是否会自动刷新缓冲区。如果取值为 false，当缓冲区溢出时就会出现异常；当 buffer 的值设为 none 时，此属性的值不能设为 false。此属性的默认值为 true。

#### 7. isThreadSafe

isThreadSafe 属性设置 JSP 页面是否可以多线程访问。如果值为 true 则此 JSP 页面可同时响应多个客户的请求；如果为 false 则在某个时刻内只能处理一个客户的请求。此属性的默认值为 true。

#### 8. info

info 属性设置 JSP 页面的信息字符串，可以是针对本 JSP 页的一些说明性文字，可用 `getServletInfo()` 方法来获得这个字符串。为什么会是 `getServletInfo()` 呢？因为 JSP 引擎实际上是把 JSP 转换为 Servlet 后再响应客户端的请求。

#### 9. errorPage

errorPage 属性设置出现异常时转向页面的 URL。

#### 10. isErrorPage

isErrorPage 属性设置是否为出错页面。如果为 true 则可以使用 exception 对象，如果为 false 则不行。默认情况下是 false，故在需要使用 exception 对象的 JSP 页面中要注意在 page 指令中设置此属性的值为 true。

### 4.7.2 include 指令

include 指令用来在该指令处静态插入一个文件，这与 C 语言中的 `#include` 类似，它只是把文件代码与本文件组合起来形成一个大的程序文件。include 指令调用的语法格式如下。

```
<%@ include file="文件路径"%>
```

文件路径一般使用相对路径，这样如果程序代码文件进行迁移也不会有所影响。路径如果以 “/” 开头，则表明使用的是相对 JSP 服务器应用的根目录路径；如果直接用文件名或是文件

夹名+文件名的形式,则表明是相对本 JSP 文件当前目录的相对路径。在包含时要保证被包含与包含文件的语法一致,使用 include 指令时要在合适的位置。

### 【实例 4-13】include 指令应用实例

本实例程序代码如下:

top.txt

```
<%@ page contentType="text/html;charset=gb2312"%>
<html>
<head>
<title>《SOA 实践者说》网站</title>
</head>
<body>
<p align="center"><b><font size="5">《SOA 实践者说》网站</font> </b></p>
```

bottom.txt

```
<%@ page contentType="text/html;charset=gb2312"%>
<hr>
<p align="center">
<font size="3">@版权所有,违者必究 2009-2012</font><br>
<font size="3">制作人:邓子云</font><br>
<font size="3">联系方式:dengziyun@126.com</font><br>
</body>
</html>
```

includeSample1.jsp

```
<%@ page contentType="text/html;charset=gb2312"%>
<%@ include file="top.txt"%>
<p align="center">
<%out.println("此书出版了,快来看看吧!还有更多的好书等着你呢!");%>
</p>
<%@ include file="bottom.txt"%>
```

本例将这三个文件放在同一目录下,所以在主文件 includeSample1.jsp 中包含其他两个文件时可直接使用文件名。在 includeSample1.jsp 文件中,第一句的 page 指令一定要放在主文件中,否则有可能部分文件的中文字符会显示为乱码;包含的第一个文件 top.txt 是文件头,包含的第二个文件 bottom.txt 是文件尾,在主文件中的是文件体,此处仅输出一句话。在实际应用中,因为同一个网站中许多网页的头部和底部是一样的,这时就可以把头部作为一个文件存放,底部作为另一个文件存放,在其他文件中只要将其包含进来即可,可大大减少程序编写及网页制作的工作量。

include 指令应用示例的运行结果如图 4-16 所示。

## 4.8 JSP 动作指令

JSP 动作指令在 JSP 程序设计中经常会用到,与 JSP 指令不同,它将影响 JSP 运行时的功能,在这一节中将对对此进行详细的介绍。





图 4-16 实例 4-13 include 指令应用示例的运行结果

### 4.8.1 include 动作指令

**include** 动作指令用来在 JSP 页面中动态包含一个文件, 这样包含页面程序与被包含页面的程序是彼此独立的, 互不影响。JSP 的 **include** 动作指令可包含一个动态文件也可以包含一个静态文件。如果包含的是一个静态文件 (如一个文本文件), 就直接输出给客户端, 由客户端的浏览器负责显示; 如果包含的是一个动态文件, 则由服务器的 JSP 引擎负责执行, 再把运行结果返回给客户端显示出来。

**注意** **include** 动作指令与 **include** 指令不同, 后者是静态包含, 将包含文件与被包含文件组合形成一个文件; 而前者是动态包含, 原理不同, 使用时也会有差别。

**include** 动作指令使用的格式如下:

```
<jsp:include page="文件路径"/>
```

或者:

```
<jsp:include page="文件路径">
  <jsp:param name="参数名 1" value="参数 1 的值"/>
  ...
  <jsp:param name="参数名 n" value="参数 n 的值"/>
</jsp:include>
```

可以看出, 在不需要传递参数时, 这两种形式的效果是一样的, 如果要传递参数就要使用第二种形式了。“文件路径”如果以 “/” 开头, 则使用相对 JSP 服务器应用的根目录路径, 如果直接用文件名或文件夹名+文件名形式, 则表明是当前目录的相对路径。

**注意** 使用时要注意被包含的是动态文件时才传递参数和参数的值。

在实例 4-13 中, 可以把主文件 `includeSample1.jsp` 改写为如下的内容, 其效果是一样的。

```
<%@ page contentType="text/html; charset=gb2312"%>
<jsp:include page="top.txt"/>
```

```
<p align="center">
  <%out.println("此书出版了，快来看看吧！还有更多的好书等着你呢！");%>
</p>
<jsp:include page="bottom.txt"/>
```

## 4.8.2 forward 动作指令

forward 动作指令用来重定向网页，即从当前网页的 forward 动作指令处转向执行另一个网页程序。forward 动作指令的调用语法格式如下：

```
<jsp:forward page="文件路径"/>
```

或者：

```
<jsp:forward page="文件路径">
  <jsp:param name="参数名 1" value="参数 1 的值"/>
  ...
  <jsp:param name="参数名 n" value="参数 n 的值"/>
</jsp:forward>
```

其中，page 参数中的值是要转向的文件，可以是相对应用服务器的路径，也可以是相对当前目录的路径；如果要转向的网页是一个动态网页，如：JSP 文件，则可以传递参数。



**注意** 页面之间的重定向也可以使用 response.sendRedirect(目的 URL)方法来实现，response 是 JSP 的内置对象，在后续章节中将会作详细介绍。

### 【实例 4-14】forward 应用程序示例

forwardExample1.jsp

```
<%@ page contentType="text/html; charset=gb2312"%>
<html>
<body>
<form name="form1" action="forwardExample1.jsp" method="post" >
程序示例链接:
<select name="goaddress" onchange="javascript:form1.submit()">
  <option value="novalue"></option>
  <option value="1">实例 4-2</option>
  <option value="2">实例 4-11</option>
  <option value="3">实例 4-14</option>
</select>
</form>
<%// forward 应用示例 Java 程序片
String s=null;
s=request.getParameter("goaddress");
if(s!=null){
  switch(s.charAt(0)){
    case '1':
%>
    <jsp:forward page="xhApp1.jsp"/>
<%
    break;
```



```

        case '2':
    %>
    <jsp:forward page="simpleCounterApp1.jsp"/>
    <%
        break;
        case '3':
    %>
    <jsp:forward page="includeSample1.jsp"/>
    <%
        break;
        default:
            out.println("你没有选择。");
    }
    }
    else
        out.println("你没有选择。");
    %>
</body>
</html>

```

在这个程序示例中，首先定义了一个表单 `form1`，表单中用一个下拉框来选择要转向的页面，这里把前面的几个实例及本实例的程序文件均放在 `Web` 应用的根目录下。在选择下拉框的选择项改变时，调用 `form1.submit()` 方法提交表单至本文件，接收到参数 `goadress` 后，根据这个字符串的第一个字符值判断要转向的页面。需要注意的是，Internet 中用 HTTP 表单提交的数据都是字符串，必要时可以用相关函数进行转换。`forward` 应用示例程序运行的结果如图 4-17 所示。



图 4-17 实例 4-14 forward 应用程序示例的运行结果

### 4.8.3 param 动作指令

在前面的 `include` 动作指令和 `forward` 动作指令中均出现了 `param` 动作指令，它用来向需要包含的动态页面或要转向的动态页面传递参数。下面来看一个传递参数的实例。

#### 【实例 4-15】param 应用程序示例

paramExample1.jsp

```

<%@ page contentType="text/html; charset=gb2312"%>
<html>
<body>

```

```

<%// param 应用示例 Java 程序片
String s=null;
s="Lest's go!";
%>
<jsp:forward page="forParam.jsp">
  <jsp:param name="s" value="<%=s%>" />
</jsp:forward>
</body>
</html>

```

#### forParam.jsp

```

<%@ page contentType="text/html; charset=gb2312"%>
<html>
<body>
<%
out.println("接收的参数 s 的值为: "+request.getParameter("s"));
%>
</body>
</html>

```

这里将两个程序文件放在同一个目录下。第一个文件 paramExample1.jsp 中的 forward 指令将转向 forParam.jsp 页面,并传递参数 s。第二个文件 forParam.jsp 输出接收的参数 s 的值。param 应用程序示例的运行结果如图 4-18 所示。



图 4-18 实例 4-15 param 应用程序示例的运行结果

### 4.8.4 useBean 动作指令

这是一个非常重要的动作指令,用来在 JSP 中创建并使用一个 JavaBean。实际工程中常用 JavaBean 做组件开发,而在 JSP 中只需要声明并使用这个组件,这样可以较大限度地实现静态内容和动态内容的分离,这也是 JSP 的优点之一。

具体的应用在后续章节中将会详细讨论 JavaBean 的开发及使用,这里重点掌握使用 JavaBean 的语法。在 JSP 中声明一个 JavaBean 的语法格式如下:

```

<jsp:useBean id="bean 的名称" scope="有效范围" class="包名.类名">
</jsp:useBean>

```

其中, id 参数是在 JSP 中这个 bean 组件的名称,只要是在它的有效范围内,均可使用这个名称来调用它; scope 为 bean 的有效范围,它的取值有四种, page、request、session、application,默认情况下取值为 page,值为 page 表示在当前 JSP 页面及当前页面所包含的静态文件中有效;



值为 `request` 表示有效范围仅在 `request` 期间；值为 `session` 表示在与每个客户的会话期间均有效；值为 `application` 表示所有客户端共享这个 `bean`，直至服务器关闭时才取消这个 `bean`。`class` 参数中如果类属于某个包则在类名的前面要加上包名，中间用“.”引用，否则直接用类名即可。

在 JSP 页面中使用 `bean` 还需要有 `import` 指令，其相关的类还必须已经作出部署，在第 8 章中将加以详细讲解，要想开发出高质量的 JSP 程序，一定要认真学习并掌握 `JavaBean` 的开发与使用。

### 4.8.5 setProperty 动作指令

这个动作指令用来设置 `Bean` 中属性的值，基本语法格式有如下四种：

```
<jsp:setProperty name="bean 的名称" property="*" />
<jsp:setProperty name="bean 的名称" property="属性名称" />
<jsp:setProperty name="bean 的名称" property="属性名称" param="参数名称" />
<jsp:setProperty name="bean 的名称" property="属性名称" value="属性值" />
```

在第一种语法格式中，`property="*"`，应用这种格式要求 `bean` 属性的名称与类型要和 `request` 对象中参数名称与类型一致，以此用 `bean` 中的属性来接收客户输入的数据，系统会根据名称来自动匹配。如果类型不一致，会根据 `bean` 中的类型进行转换。



**注意** 由于用户输入的数据往往不规范，数据类型的转换可能会出错，因此在程序中要及时捕获并报告错误，以增强程序的健壮性。

第二种语法格式则只设置其中匹配的一个 `bean` 的属性。

第三种语法格式根据指定的 `request` 对象中的参数与属性匹配。



**注意** 如果在 `request` 中有空值，或根据名称 `bean` 中的属性找不到与 `request` 参数相匹配的参数，则都不会在这个属性中设置任何值。

第四种语法格式用来给 `bean` 的属性赋值，属性值的数据类型要与属性的数据类型一致，否则程序会出错，因此要进行数据类型的转换。JSP 中提交的数据一般是字符串，故要将字符串转换为所需的数据类型。字符串转换为其他数据类型的函数分别如下。

- (1) 转换为 `boolean`，方法为 `Boolean.valueOf(String str).booleanValue()`。
- (2) 转换为 `Boolean`，方法为 `Boolean.valueOf(String str)`。
- (3) 转换为 `byte`，方法为 `Byte.valueOf(String str).byteValue()`。
- (4) 转换为 `Byte`，方法为 `Byte.valueOf(String str)`。
- (5) 转换为 `char`，方法为 `Character.valueOf(String str).charValue()`。
- (6) 转换为 `Character`，方法为 `Character.valueOf(String str)`。
- (7) 转换为 `double`，方法为 `Double.valueOf(String str).doubleValue()`。
- (8) 转换为 `Double`，方法为 `Double.valueOf(String str)`。
- (9) 转换为 `int`，方法为 `Integer.valueOf(String str).intValue()`。

- (10) 转换为 Integer, 方法为 Integer.valueOf(String str)。
- (11) 转换为 float, 方法为 Float.valueOf(String str).floatValue()。
- (12) 转换为 Float, 方法为 Float.valueOf(String str)。
- (13) 转换为 long, 方法为 Long.valueOf(String s).longValue()。
- (14) 转换为 Long, 方法为 Long.valueOf(String s)。

这些方法在 JSP 编程中都是常用的, 读者要理解并掌握它们。

**注意** 在同一个 setProperty 动作指令中不能同时存在 param 和 value 参数。

setProperty 动作指令可以在 useBean 动作指令中使用, 也可在声明了 useBean 后使用, 但不能在声明之前使用。与 userBean 动作指令结合使用的格式如下。

```
<jsp:useBean id="bean 的名称" scope="有效范围" class="包名.类名">
    <jsp:setProperty name="bean 的名称" property="属性名称" value="属性值"/>
    ...
    <jsp:setProperty name="bean 的名称" property="属性名称" value="属性值"/>
</jsp:useBean>
```

### 4.8.6 getProperty 动作指令

getProperty 动作指令用来获得 bean 的属性并将其转换为字符串, 再在 JSP 页面中输出。使用的语法格式如下。

```
<jsp:getProperty name="bean 的名称" property="属性名称"/>
```

## 4.9 JSP 中的中文字符处理

JSP 程序员在编写 JSP 程序时, 常常会碰到处理中文字符的问题, 在接受 request 的中文字符时显示出来却是乱码, 一起来看看如何处理这个问题。

在 JSP 中客户提交的含有汉字的数据必须采用特殊的处理方式, 先将得到的字符串用 ISO-8895-1 编码, 并放到一个字节数组中, 再用 String 类的构造函数将其转换为字符串对象。这个过程的程序代码如下。

```
String tempString=request.getParameter("name");
byte tempB[ ]=tempString.getBytes("ISO-8859-1");
tempString=new String(tempB);
```

上述代码中, 用 tempString 接收用户提交的参数 name, 将其转换为一个 byte 数组后再用 String 的构造函数重新生成字符串, 这样将不会再显示为乱码。可以将上面的码编写一个字符串处理函数, 用一个静态文件保存, 在需要处理中文字符的 JSP 页面中将其包含进来, 就可以在 JSP 页中使用这个函数了。这个函数的代码如下。

```
public String codeToString(String str){//处理中文字符串的函数
    String s=str;
```



```

    try{
        byte tempB[]=s.getBytes("ISO-8859-1");
        s=new String(tempB);
        return s;
    }catch(Exception e){
        return s;
    }
}

```

### 【实例 4-16】中文字符处理程序示例

chineseStringExample1.jsp

```

<%@ page contentType="text/html;charset=gb2312"%>
<%!
public String codeToString(String str){//处理中文字符串的函数
    String s=str;
    try{
        byte tempB[]=s.getBytes("ISO-8859-1");
        s=new String(tempB);
        return s;
    }catch(Exception e){
        return s;
    }
}
%>
<html>
<body>
<form name="form1" action="chineseStringExample1.jsp" method="post">
请输入你的姓名:
<input type="text" name="username">
<input type="submit" name="submit" value="提交">
</form><br>
<%
    if(request.getParameter("username")==null)
        out.println("你没有输入姓名。");
    else
        out.println("你输入的姓名为: "+codeToString(request.getParameter ("username")));
%>
</body>
</html>

```

在程序代码中，“<%!”与“%>”声明的是变量和方法，在此声明了一个表单。其中有一个文本框，名为 username，用来让用户输入姓名。方法 codeToString()将提交的姓名进行转换以确保正确输出。

在使用中文字符串时，一定要注意小心转换，以免出现乱码时不知所措。其示例程序代码的运行结果如图 4-19 所示。



图 4-19 实例 4-16 中文字符处理程序示例的运行结果

## 4.10 小结

掌握 JSP 语法是学好 JSP 开发的基础。JSP 网页包括 HTML 标签和 Java 程序片，因此在 Java 中使用的各种数据类型在 JSP 中均可以使用。Java 的数据类型分为简单数据类型和复合数据类型，简单数据类型包括整数、实数、布尔数及字符等，是不能再简化、系统已内置的数据类型；复合数据类型由简单数据类型组合而成，如：类、接口等。

Web 开发中传递的数据大多是字符串，因此在 `String` 类、`StringBuffer` 类及其方法中，要重点掌握字符串与其他数据类型的转换。`String` 类表示的字符串是定长的，而 `StringBuffer` 类提供了可变长的字符串。

Java 中的运算符特别丰富，分为算术运算符、关系运算符、布尔运算符和移位运算符。在 Java 中控制逻辑分为选择分支、循环和跳转三种，Java 程序片用“`<%`”和“`%>`”括起来。在程序中加入注释可增强程序的可读性与可理解性，JSP 中的注释又分为 HTML 注释和 JSP 注释两种。

JSP 指令有 `page` 指令和 `include` 指令两种。`page` 指令定义 JSP 页面的全局属性；`include` 指令用来在该指令处静态插入一个文件。

JSP 的动作指令较多，有 `include`、`forward`、`param`、`useBean`、`setProperty`、`getProperty`。`include` 动作指令用来在 JSP 页面中动态包含一个文件，包含的文件可以是动态文件也可以是静态文件。`forward` 动作指令用来重定向网页；`param` 动作指令配合 `include` 动作指令和 `forward` 动作指令使用，用来设置这两个动作指令的参数值；`useBean` 在 JSP 中创建并使用一个 `JavaBean`；`setProperty` 和 `getProperty` 分别用来设置和获得 `bean` 的属性值。

中文字符处理是 JSP 程序员常常感到头痛的问题，如果把获得的字符串转换为一个 `byte` 数组再转换为字符串，就可以正确地显示中文了。

## 4.11 练习

1. 编写一个简单的网站计数器（程序代码可参考本章的实例 4-12）。
2. 编写一个中文字符转换函数。
3. 试比较说明 `include` 指令与 `include` 动作指令的区别。



# 05

## JSP 的内置对象

---

JSP 的内置对象在 JSP 页面中无须声明就可以直接使用，其中内置对象常用的有：`request`、`response`、`session`、`application`、`out`、`config`、`pageContext`，在本章中将一一介绍并以实例指引开发，读者重点要掌握 `request`、`response`、`session`、`application`、`out` 五个对象的应用。

学完本章后，读者应对 JSP 的内置对象有较为深刻的认识，并能在程序开发及工程实践中予以运用。

### 5.1 内置对象概述

JSP 的内置对象包括 `request`、`response`、`session`、`application`、`out`、`config`、`pageContext`。这些对象在服务器端和客户端交互的过程中分别完成不同的功能。

#### 1. request

`request` 对象与 `HttpServletRequest` 类关联，是 `javax.servlet.ServletRequest` 的一个子类。用 `request` 对象可以获取客户端提交的数据，如：表单中的数据、网页地址后带的参数等。

#### 2. response

`response` 对象与 `HttpServletResponse` 类关联，可用来向客户端输入数据。

#### 3. session

`session` 对象与 `HttpSession` 类关联，可用来保存在服务器与一个客户端之间需要保留的数据，当客户端关闭网站（或称为系统）的所有网页时，`session` 变量会自动清除。HTTP 是一个无状态的协议，不保留会话间的数据，通过 `session` 对象扩展了 HTTP 的功能。



**注意** 如果使用 `page` 指令关闭了 `session`，则这些页面中调用 `session` 对象将导致错误，因此在工程实践中，一般不必关闭 `session`。

`request`、`response`、`session` 是 JSP 内置对象中重要的三个对象，这三个对象提供了服务器端与客户端（即浏览器）进行交互通信的控制，它们的控制如图 5-1 所示。

当客户端打开浏览器时，在地址栏中输入服务器 Web 服务页面的地址后，就会显示 Web 服务器上的网页。客户端的浏览器从 Web 服务器上获得网页实际上是使用 HTTP 协议，向服务器端发送了一个请求，服务器在收到来自客户端浏览器发来的请求后响应该请求。JSP 通过 `request` 对象控制客户浏览器的请求，通过 `response` 对客户浏览器进行响应。而 `session` 就保存这个会话期间需要使用的数据信息。

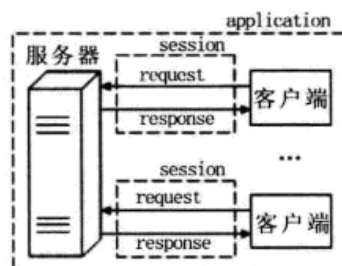


图 5-1 JSP 的三个重要内置对象

请求的方法有许多种，有 `post`、`get`、`put`、`head`、`delete`、`trace` 等，最常用的是前两种。`request` 和 `response` 对象还可以操作 HTTP 头中的数据。

#### 4. application

`application` 对象与 `ServletContext` 类关联，一旦创建（在服务器开始提供服务时，即第一次被访问时 `application` 对象就会被创建），就会一直保持直到服务器关闭服务为止。所以 `application` 对象可用来提供一些全局的数据和对象。

#### 5. out

`out` 对象实际上是使用 `PrintWriter` 类来向客户端浏览器输出数据。

#### 6. config

`config` 对象是 `ServletConfig` 类的一个对象，是 JSP 配置处理程序的句柄，在 JSP 页面范围内有效。

#### 7. pageContext

`pageContext` 用来管理属于 JSP 中特殊可见部分中已命名对象的访问。

## 5.2 request 对象

`request` 对象封装了客户端提交的数据信息，包括用户提交的信息以及客户端的一些信息。客户端可通过 HTML 表单或者在网页地址后面使用参数的方法提交数据，再用 `request` 对象的相关方法来获取提交的各种数据。

### 5.2.1 request 对象的方法

`request` 对象所属的类实现了 `javax.servlet.http.HttpServletRequest` 接口，此接口位于 `servlet-api.jar` 包中，`javax.servlet.http.HttpServletRequest` 接口的声明如下。

```
public abstract interface javax.servlet.http.HttpServletRequest
    extends javax.servlet.ServletRequest;
```



可见 `javax.servlet.http.HttpServletRequest` 扩展自 `javax.servlet.ServletRequest`。`request` 对象的常用方法有如下一些。

### 1. `getAttribute()`

此方法将参数 `name` 指定的属性值作为一个 `Object` 对象返回；如果参数 `name` 中给定的名字不存在相应的属性则返回 `null`，调用方法如下：

```
public Object getAttribute(String name)
```

### 2. `getAttributeNames()`

此方法得到一个 `Enumeration`（枚举型）对象，其中的值为此请求中可用的属性的名称，调用方法如下：

```
public java.util.Enumeration getAttributeNames()
```

### 3. `getCharacterEncoding()`

此方法返回请求中所用的字符编码的名称，如果未指定字符编码则返回 `null`，调用方法如下：

```
public String getCharacterEncoding()
```

### 4. `getContentType()`

此方法返回请求的 `MIME` 类型，如果类型未知则返回 `null`，调用方法如下：

```
public String getContentType()
```

### 5. `getContextPath()`

此方法得到请求 `URI` 中请求的应用上下文路径。如果当前 `Web` 应用就是 `Tomcat` 安装目录的 `webapps` 子目录中的 `ROOT`，则 `getContextPath()` 方法会返回 `""`；如果为其他目录，则返回相对于 `Web` 应用服务器的根目录的子目录。如果当前 `Web` 应用为 `webapp`，则 `getContextPath()` 方法的返回值为 `"/webapp"`，调用方法如下：

```
public String getContextPath()
```

### 6. `getLocalAddr()`

此方法返回接收请求的 `Web` 服务器的地址，调用方法如下：

```
public String getLocalAddr()
```

### 7. `getLocalName()`

此方法返回接收请求的 `Web` 服务器的机器名，调用方法如下：

```
public String getLocalName()
```

### 8. `getMethod()`

此方法得到请求所用的 `HTTP` 请求类型，如 `GET`、`POST`、`PUT` 等，调用方法如下：

```
public String getMethod()
```

### 9. getParameter()

此方法得到请求中指令的参数值，其结果作为 `String`，如果该参数不存在则会返回 `null`，调用方法如下：

```
public String getParameter(String name)
```

### 10. getParameterMap()

此方法得到请求中所有参数的一个 `Map` 对象，其中参数名为键，字符串数组作为值，调用方法如下：

```
public java.util.Map getParameterMap()
```

### 11. getParameterNames()

此方法获得所有请求中参数名称的枚举，调用方法如下：

```
public java.util.Enumeration getParameterNames()
```

### 12. getParameterValues()

此方法得到所有请求参数值的数组，数组的内容为请求中指定参数 `name` 的多个值。如果在请求中找不到 `name` 参数则返回 `null`，调用方法如下：

```
public String[] getParameterValues(String name)
```

### 13. getProtocol()

此方法返回请求所用协议的名称和版本，其形式为：协议名称/主版本号.副版本号，如 `HTTP/1.1`，调用方法如下：

```
public String getProtocol()
```

### 14. getQueryString()

此方法返回请求 `URI` 中所包含的位于路径之后的查询字符串，调用方法如下：

```
public String getQueryString()
```

### 15. getRemoteAddr()

此方法返回发送请求的客户端 `IP` 地址，调用方法如下：

```
public String getRemoteAddr()
```

### 16. getRemoteHost()

此方法得到发送请求的客户端机器名，如果无法获得机器名则返回 `IP` 地址，调用方法如下：

```
public String getRemoteHost()
```

### 17. getRemotePort()

此方法获得客户端或发送请求的最后一个代理的端口号，调用方法如下：

```
public String getRemotePort()
```



### 18. getRequestURI()

此方法返回请求路径 URI，调用方法如下：

```
public String getRequestURI()
```

### 19. getRequestMethod()

此方法得到请求 URL，包括协议、服务器名、端口号和 URI 路径，调用方法如下：

```
public StringBuffer getRequestMethod()
```

### 20. getServerPort()

此方法得到接收请求的服务器端的端口号，调用方法如下：

```
public int getServerPort()
```

### 21. getServletPath()

此方法获得当前请求 URI 中标识 Servlet 的部分，如果是 JSP 页面则是 JSP 页面的完整上下文路径，调用方法如下：

```
public String getServletPath()
```

### 22. getSession()

此方法得到与当前请求相关联的 HttpSession 对象，如果此请求还没有会话则创建一个新的 HttpSession 对象并返回这个对象，调用方法如下：

```
public HttpSession getSession()
```

### 23. removeAttribute()

此方法从请求中删除指定名称 name 的属性，调用方法如下：

```
public void removeAttribute(String name)
```

### 24. setAttribute()

此方法将指定的属性对象 attribute 用名称 name 保存于请求中，调用方法如下：

```
public Object setAttribute(String name, Object attribute)
```

### 25. setCharacterEncoding()

此方法设置读取请求数据时所使用的字符编码，此方法必须在调用 request 的所有其他方法之前调用，其后才能再行读取参数，调用方法如下：

```
public void setCharacterEncoding(String encoding)
```

## 【实例 5-1】request 常用方法的应用

requestApp1.jsp

```
<%@ page contentType="text/html; charset=gb2312"%>
<%@ page import="java.util.*" %>
```

```

<html>
<head>
</head>
<body>
请求信息如下:<br>
请求的方法是:<%=request.getMethod()%>
<br>
请求的 URI 是:<%=request.getRequestURI()%>
<br>
请求的协议是:<%=request.getProtocol()%>
<br>
接受客户提交信息的页面路径是:<%=request.getServletPath()%>
<br>
请求的协议是:<%=request.getProtocol()%>
<br>
请求中的查询字符串是:<%=request.getQueryString()%>
<br>
请求信息的总长度是:<%=request.getContentLength()%>
<br>
服务器名称是:<%=request.getServerName()%>
<br>
提供 HTTP 服务的服务器端口号是:<%=request.getServerPort()%>
<br>
客户端 IP 地址是:<%=request.getRemoteAddr()%>
<br>
客户端机器的名称是:<%=request.getRemoteHost()%>
<br>
HTTP 头文件中 User-Agent 的值是: <%=request.getHeader("User-Agent")%>
<br>
HTTP 头文件中 accept 的值是: <%=request.getHeader("accept")%>
<br>
HTTP 头文件中 Host 的值是: <%=request.getHeader("Host")%>
<br>
HTTP 头文件中 accept-encoding 的值是: <%=request.getHeader("accept- encoding")%>
<br>头名字的一个枚举:
    <% Enumeration enumHead=request.getHeaderNames();
        while(enumHead.hasMoreElements())
            out.println((String)enumHead.nextElement());
    %>
</body>
</html>

```

该程序代码用 `request` 的各种方法输出 `request` 中的各种信息。在程序后面用到了 `Enumeration` 类, 这个类在 `java.util` 包中, 故在页面的第二句用 `<%@ page import="java.util.*" %>` 语句把这个包导入, 这样才能在后续程序代码中使用 `Enumeration` 类, 否则会出错, 得到相应的枚举类实例后, 用一个循环语句把实例中的所有枚举一一输出在浏览器中。运行结果中得到的客户端 IP 地址是 127.0.0.1, 这里因为是在本地调试, 127.0.0.1 代表本地计算机, 故得到的是此 IP 地址。其程序代码运行的结果如图 5-2 所示。





图 5-2 实例 5-1 request 常用方法的应用程序运行结果

## 5.2.2 获得表单数据

在 JSP 中，服务器端程序与客户端交互最常用的方法就是采用表单提交数据，HTML 表单的内容在 3.1 节中已做了较为详细的介绍。表单提交的方法主要有两种，一种是 **get** 方法，另一种是 **post** 方法，两者最大的区别是：使用 **get** 方法提交的数据会显示在浏览器的地址栏中，而 **post** 方法则不会显示，故 **post** 方法更为常用。表单中提交的数据可以是文本框、列表框及文本区域等。使用 **request** 对象的 **getParameter()** 方法可得到表单中相应数据项的值。

### 【实例 5-2】获得表单数据

#### userRegist1.jsp

```
<%@ page contentType="text/html; charset=gb2312"%>
<script language="javascript">
function on_submit(){//验证数据的合法性
    if (form1.username.value == ""){
        alert("用户名不能为空，请输入用户名！");
        form1.username.focus();
        return false;
    }
    if (form1.userpassword.value == ""){
        alert("用户密码不能为空，请输入密码！");
        form1.userpassword.focus();
        return false;
    }
    if (form1.reuserpassword.value == ""){
        alert("用户确认密码不能为空，请输入密码！");
        form1.reuserpassword.focus();
        return false;
    }
}
```

```

        if (form1.userpassword.value != form1.reuserpassword.value){
            alert("密码与确认密码不同");
            form1.userpassword.focus();
            return false;
        }
        if (form1.email.value.length!= 0){
            for (i=0; i<form1.email.value.length; i++){
                if (form1.email.value.charAt(i)=="@") break;
                if (i==form1.email.value.length){
                    alert("非法 E-mail 地址! ");
                    form1.email.focus();
                    return false;
                }
            }
        }else{
            alert("请输入 E-mail! ");
            form1.email.focus();
            return false;
        }
    }
}
</script>
<html>
<head>
<title>新用户注册</title>
</head>
<body>
<form method="POST" action="acceptUserRegist1.jsp" name="form1" onsubmit="return
on_submit()">
    新用户注册<br>
    用户名(*): <input type="text" name="username" size="20"><br>
    密    码(*): <input type="password" name="userpassword" size="20"><br>
    再输一次密码(*): <input type="password" name="reuserpassword" size="20"><br>
    性别: <input type="radio" value="男" checked name="sex">男<input type="radio"
name="sex" value="女">女<br>
    出生年月: <input name="year" size="4" maxlength=4>年
        <select name="month">
            <option value="1" selected>1</option>
            <option value="2">2</option>
            <option value="3">3</option>
            <option value="4">4</option>
            <option value="5">5</option>
            <option value="6">6</option>
            <option value="7">7</option>
            <option value="8">8</option>
            <option value="9">9</option>
            <option value="10">10</option>
            <option value="11">11</option>
            <option value="12">12</option>
        </select>月
        <input name="day" size="3" maxlength=4>日<br>
    电子邮箱(*): <input name="E-mail" maxlength=28><br>
    家庭住址: <input type="text" name="address" size="20"><br>
    <input type="submit" value="提交" name="B1"><input type="reset"
value="全部重写" name="B2"><br>
</form>
</body>
</html>

```



## acceptUserRegist1.jsp

```

<%@ page contentType="text/html; charset=gb2312"%>
<%!
public String codeToString(String str){//处理中文字符串的函数
    String s=str;
    try{
        byte tempB[]=s.getBytes("ISO-8859-1");
        s=new String(tempB);
        return s;
    }catch(Exception e){
        return s;
    }
}
%>
<html>
<head>
<title>接收新用户注册</title>
</head>
<body>
这是新用户注册所提交的数据:
<br>
用户名是: <%=codeToString(request.getParameter("username"))%><br>
密码是: <%=codeToString(request.getParameter("userpassword"))%><br>
性别是: <%=codeToString(request.getParameter("sex"))%><br>
出生年月是:
<%=request.getParameter("year")+request.getParameter("month")+request.
getParameter("day")%><br>
电子邮箱是: <%=request.getParameter("E-mail")%><br>
家庭住址是: <%=codeToString(request.getParameter("address"))%><br>
</body>
</html>

```

userRegist1.jsp 页面是一个新用户注册表单, 将表单数据提交给 acceptUserRegist1.jsp 页面, 这两个页面放在同一个目录下。

在 userRegist1.jsp 页面中, 仍然用 JavaScript 语句验证数据的合法性, 方法同实例 5-2, 验证通过后才提交数据。此页面的运行结果如图 5-3 所示。

新用户注册

用户名(\*):

密码(\*):

再输一次密码(\*):

性别: ☐ 男 ☒ 女

出生年月: 2005 年 6 月 4 日

电子邮箱(\*):

家庭住址:

图 5-3 用户注册页面

在 `acceptUserRegist1.jsp` 页面中,为了解决中文显示的问题,加入了一个中文字符处理函数,在接收用户名、密码及家庭住址等参数时均用到了此函数,用 `request` 对象的 `getParameter()` 方法获取相应表单项的数据,此方法中的参数是表单中相应数据项的名称。这个页面在 `userRegist1.jsp` 提交数据后的运行结果如图 5-4 所示。



图 5-4 新用户注册提交的数据

程序中,数据验证与显示由客户端浏览器来完成,JavaScript 语句由客户端浏览器负责解释,在接收正确的数值后,再用 JSP 的程序来做出相应的处理。

## 5.3 response 对象

客户端浏览器每访问一次 Web 服务器的页面都会提交一次请求,与 `request` 对象对应的是 `response` 对象,该对象可以用来对客户请求做出响应,向客户端发送数据。输出的数据可以是各种数据类型,甚至是文件,这可以通过 `page` 指令的 `contentType` 属性或是 `response` 的 `setContentType()` 方法来设置。

### 5.3.1 response 对象的方法

`response` 对象代表对客户端的响应,它实现了 `javax.servlet.ServletResponse` 接口,此接口位于 `servlet-api.jar` 包中,这个接口的声明情况如下:

```
public abstract interface javax.servlet.http.HttpServletResponse
    extends javax.servlet.ServletResponse;
```

可见 `javax.servlet.http.HttpServletResponse` 又扩展自 `javax.servlet.ServletResponse`。

`response` 对象的常用方法有如下一些。

#### 1. flushBuffer()

此方法强制将 `response` 缓冲区中的所有内容写到客户端,调用方法如下:

```
public void flushBuffer() throws IOException
```

#### 2. getBufferSize()

此方法得到 `response` 的实际缓冲区大小,单位为字节,如果没有使用缓冲区则返回 0,调



用方法如下：

```
public int getBufferSize()
```

### 3. getCharacterEncoding()

此方法得到当前 **response** 中的字符集名称，调用方法如下：

```
public String getCharacterEncoding()
```

### 4. getContentType()

此方法获得 **response** 中所发送的 **MIME** 类型，调用方法如下：

```
public String getContentType()
```

### 5. getOutputStream()

此方法返回一个可用于在 **response** 中写入二进制数据的输出流 **ServletOutputStream**。在 JSP 页面中建议不使用这个方法，因为 JSP 页面中的数据为文本数据，调用方法如下：

```
public ServletOutputStream getOutputStream() throws IOException
```

### 6. getWriter()

此方法返回一个可用于在 **response** 中写入文本数据的 **PrintWriter** 对象。在 JSP 页面中建议不使用这个方法，因为它可能会影响 Web 容器的一些内在机制，调用方法如下：

```
public PrintWriter getWriter() throws IOException
```

### 7. sendRedirect()

此方法用于进行页面重定向，参数 **location** 要导向目标地址，可以是相对路径也可以是绝对路径，调用方法如下：

```
public void sendRedirect() (String location) throws IOException
```

### 8. setBufferSize()

此方法设置 **response** 的响应缓冲区大小（字节数），调用方法如下：

```
public void setBufferSize() (int size)
```

### 9. setCharacterEncoding()

此方法设置 **response** 的响应字符编码，也就是 **Content-Type** 中的 **charset** 属性值，此方法必须在调用 **getWriter()** 方法前调用，调用方法如下：

```
public void setCharacterEncoding() (String encoding)
```

### 10. setContentType()

此方法设置 **response** 的 **Content-Type** 类型，如果其中包括了 **charset** 属性值，则会设置字符编码，此方法必须在调用 **getWriter()** 方法前调用，调用方法如下：

```
public void setContentType() (String type)
```

### 5.3.2 使用 Cookie

Cookie 是服务器发送给客户端浏览器的体积较小的纯文本信息，以后当用户访问同一个 Web 服务器时，浏览器会把它们发送给服务器。通过让服务器读取原先保存在客户端的信息，为浏览者提供一系列的方便。例如在线交易过程中标识用户身份，在安全要求不高的场合可避免用户重复输入名字和密码、门户网站的主页定制、有针对性地投放广告等。

使用 Cookie 可以为用户带来方便，但在安全性要求较高的场合下，建议不要使用 Cookie。此外，浏览器中只允许存放 300 个 Cookie，每个站点最多存放 20 个 Cookie，每个 Cookie 的大小限制为 4 KB，故不必担心 Cookie 会过多地占用硬盘的空间。

Cookie 存放在客户端，首先要新建一个 Cookie，然后设置其属性，再通过 response 对象的 addCookie() 方法将其放入客户端，获取 Cookie 对象可调用 request 对象的 getcookies() 方法。

#### 1. 创建 Cookie

调用 Cookie 对象的构造函数可以创建 Cookie，构造函数的语法格式如下：

```
Cookie(String cookieName,String cookieValue);
```

在函数中，第一个参数 cookieName 是新建的 Cookie 对象的名称，第二个参数 cookieValue 是新建的 Cookie 对象的值。



**注意** Cookie 对象的名称和值都不能包含空白字符以及下列字符：

[ ] ( ) = , " / ? @ :

#### 2. 设置与读取 Cookie 属性

在把 Cookie 加入待发送的应答头之前，可以查看或设置 Cookie 的各种属性。

getComment()/setComment(): 获取/设置 Cookie 的注释。

getDomain()/setDomain(): 获取/设置 Cookie 适用的域。

使用这两个方法可以指示浏览器把 Cookie 返回给同一域内的其他服务器。注意域必须以点开始（例如.sitename.com），非国家类的域（如.com，.edu，.gov）必须包含两个点，国家类的域（如.com.cn，.edu.uk）必须包含三个点，一般的，Cookie 只返回与发送它的服务器名称完全相同的服务器。

getMaxAge()/setMaxAge(): 获取/设置 Cookie 过期之前的时间，以秒计。如果不设置该值，则 Cookie 只在当前会话内有效，即在用户关闭浏览器之前有效，而且这些 Cookie 不会保存到磁盘上。

getName()/setName(): 获取/设置 Cookie 的名称。本质上，名称和值是需要程序员始终关心的两个部分。由于 HttpServletRequest 的 getCookies 方法返回的是一个 Cookie 对象的数组，因此通常要用循环语句来访问这个数组并查找特定名称，然后用 getValue() 检查它的值。

getPath()/setPath(): 获取/设置 Cookie 适用的路径。如果不指定路径，Cookie 将返回给当前页面所在目录及其子目录下的所有页面。这里的方法可以用来设定一些条件。例如，



`someCookie.setPath("/")`，此时服务器上的所有页面都可以接收到该 Cookie。

`getSecure()/setSecure()`：获取/设置一个 `boolean` 值，该值表示 Cookie 是否只能通过加密的连接（即 SSL）发送。

`getValue()/setValue()`：获取/设置 Cookie 的值。如前所述，名称和值实际上是程序员始终关心的两个方面。不过也有一些例外情况，比如把名称作为逻辑标记（也就是说，如果该名称存在，则表示 `true`）。

`getVersion()/setVersion()`：获取/设置 Cookie 所遵从的协议版本。默认版本 0 遵从原先的 Netscape 规范；版本 1 遵从 RFC 2109，但尚未得到广泛的支持。

### 3. 将 Cookie 加入 HTTP 头中

可将 `addCookie()` 方法加入到 Set-Cookie 应答头，如下所示。

```
Cookie usernameCookie = new Cookie("username", "dzycsai");
response.addCookie(usernameCookie);
```

### 4. 读取 Cookie

从客户端读取 Cookie 时调用的是 `request` 对象的 `getCookies()` 方法。该方法返回一个与 HTTP 请求头中的内容对应的 Cookie 对象数组。得到这个数组之后，一般是用循环语句访问其中的各个元素，并调用 `getName()` 检查各个 Cookie 的名字，直至找到目标 Cookie 为止；然后对这个目标 Cookie 调用 `getValue()`，根据获得的结果进行其他处理。

### 5. Cookie 工具函数

用 `request` 对象的 `getCookies()` 方法得到一个 Cookie 对象的数组，要找到指定的 Cookie 对象比较麻烦，这时可以编写一个得到指定名称的 Cookie 对象值的方法，这里，提供一个这样的函数。

```
String getCookieValue(Cookie[] cookieArray,
    String cookieName, String defaultValue) {
    for(int i=0; i<cookieArray.length; i++){
        Cookie cookie = cookieArray[i];
        if (cookieName.equals(cookie.getName()))
            return(cookie.getValue());
    }
    return(defaultValue);
}
```

在此方法中，第一个参数是一个 Cookies 数组，可以用 `request.getCookies()` 方法得到；第二个参数 `cookieName`，查找 Cookie 对象的名称；第三个参数 `defaultValue` 是在没有找到指定的 Cookie 对象时返回的值。

#### 5.3.3 response 对象重定向

在 JSP 程序设计中，经常要进行页面的重定向，如在页面程序中加入判断语句即满足条件就转向某页面，不满足条件则转向另一个页面。看下面的一个程序实例。

## 【实例 5-3】页面重定向程序示例

sendRedirectExample1.jsp

```

<%@ page contentType="text/html; charset=gb2312"%>
<html>
<body>
<%//页面重定向程序片
String url;
url=request.getParameter("goaddress");
if(url!=null) response.sendRedirect(url);
%>
<form name="form1" action="sendRedirectExample1.jsp" method="post" >
页面重定向:
<select name="goaddress" onchange="javascript:form1.submit()">
<option value="">=====请选择=====</option>
<option value="http://www.hnwlw.net">湖南省物流公共信息平台</option>
<option value="http://www.56edu.com">湖南现代物流职业技术学院</option>
<option value="http://www.hnii.gov.cn">湖南省信息化工作办公室</option>
<option value="http://www.temco.com.cn">天工远科信息技术有限公司</option>
</select>
</form>
</body>
</html>

```

在此程序中，声明了一个表单，把数据提交给本页，当改变下拉选择框的选项时会提交表单；在程序片中，接收到提交的表单中要转向的地址后，用 `response` 对象的 `sendRedirect()` 方法进行页面的重定向；在程序片中要判断接收到的参数是否为空，因为数据提交给本页面，当表单数据没有提交时，得到相应数据项的数据会为空，此时并不进行页面的重定向。此程序的运行结果如图 5-5 所示。



图 5-5 实例 5-3 页面重定向程序示例的运行结果

## 5.3.4 定时刷新页面

在许多时候需要定时刷新网页，如刷新时间。看下面一个程序实例。

## 【实例 5-4】定时刷新页面程序示例

refreshExample1.jsp

```

<%@ page contentType="text/html; charset=gb2312"%>

```



```

<%@ page import="java.util.Date"%>
<html>
<head>
<title>定时刷新页面</title>
</head>
<body>
<%//设置刷新页面的时间，每隔 1 秒钟刷新一次
response.setHeader("refresh","1");
%>
当前的系统时间是:
<%//输出当前最新的时间
out.println(new Date());
%>
</body>
</html>

```

该程序中用到了有关日期的方法，可在页面开始处用 `import` 指令导入 `java.util.Date` 类；程序通过 `response` 对象的 `setHeader()` 方法设置 HTTP 头中 `refresh` 信息的值，使得网页不断刷新从而得到当前最新的时间，实现了每隔 1 秒钟刷新页面。该程序的运行结果如图 5-6 所示。

这个程序显示的月份和日期均是美国习惯的表示法，下面对这段代码稍作修改，以适合中国人的使用习惯。

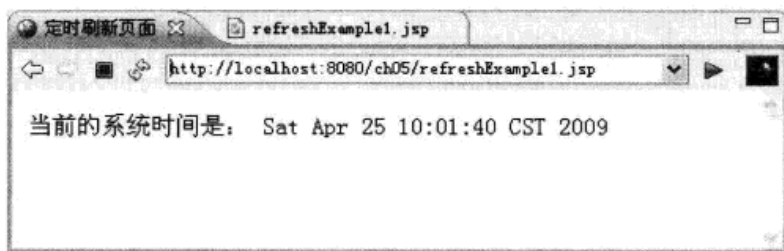


图 5-6 实例 5-4 定时刷新页面程序示例的运行结果

#### refreshExample2.jsp

```

<%@ page contentType="text/html; charset=gb2312"%>
<%@ page import="java.util.Date"%>
<html>
<head>
<title>定时刷新页面</title>
</head>
<body>
<%//设置刷新页面的时间，每隔 1 秒钟刷新一次
response.setHeader("refresh","1");
%>
当前的系统时间是:
<%//输出当前最新的时间
    Date thisDay=new Date();
    SimpleDateFormat sdf=
        new SimpleDateFormat("yyyy'年'MM'月'dd'日' hh'时'mm'分'ss'秒'");
    out.print(sdf.format(thisDay));
%>
</body>
</html>

```

这里采用了 `SimpleDateFormat` 类来格式化日期型数据的显示格式, `SimpleDateFormat` 类的构造函数参数为格式化字符串, 即 “yyyy'年'MM'月'dd'日' hh'时'mm'分'ss'秒'”。程序运行的结果如图 5-7 所示。

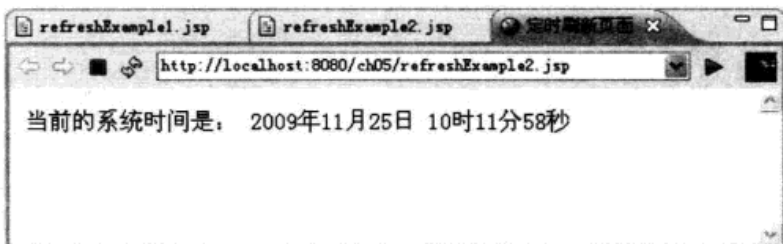


图 5-7 时间改用符合习惯的表达形式

## 5.4 session 对象

`session` 对象用来保存一些在与每个用户会话期间需要保持的数据信息, 这样就方便了会话期间的一些处理程序。如可以用 `session` 变量记住用户的用户名, 以后就不必在其他的网页中重复输入了。`session` 对象的信息保存在服务器中, 但 ID 保存在客户机的 Cookie 中, 如果客户机不支持 Cookies 则转为 URL 重写, 一般在使用 `session` 对象时不必考虑其实现的细节问题。

当用户关闭了所有某个 Web 服务器上的网页时, 则此服务器与客户机的 `session` 对象以及其变量会自动消失。

### 5.4.1 session 对象的方法

`session` 对象代表客户端与服务端的会话, 实现了 `javax.servlet.http.HttpSession` 接口, 此接口位于 `servlet-api.jar` 包中。`session` 对象的常用方法有如下一些。

#### 1. `getAttribute()`

此方法得到会话中指定的名称 `name` 相关的对象, 如果找不到则返回 `null`, 调用方法如下:

```
public Object getAttribute (String name)
```

#### 2. `getAttributeNames()`

此方法返回会话中所有对象名称的枚举, 调用方法如下:

```
public java.util.Enumeration getAttributeNames()
```

#### 3. `getMaxInactiveInterval()`

此方法客户端会话有效时间间隔, 即如果客户在网站上什么事也不做还能保持会话有效的时间间隔, 单位为秒, 调用方法如下:

```
public int getMaxInactiveInterval()
```



#### 4. getServletContext()

此方法返回当前会话所属的 `ServletContext`，调用方法如下：

```
public javax.servlet.ServletContext getServletContext()
```

#### 5. invalidate()

此方法使当前会话失效，调用方法如下：

```
public void invalidate()
```

#### 6. isNew()

此方法如果服务器端尚未接受过客户端当前会话的请示，则此方法返回 `true`，否则返回 `false`，调用方法如下：

```
public boolean isNew()
```

#### 7. removeAttribute(String name) ()

此方法删除会话中指定名称 `name` 所代表的对象，调用方法如下：

```
public void removeAttribute(String name)
```

#### 8. setMaxInactiveInterval()

此方法设置会话有效的时间间隔，即如果客户在网站上什么事也不做还能保持会话有效的时间间隔，单位为秒，调用方法如下：

```
public void setMaxInactiveInterval()
```

### 【实例 5-5】记住会话的用户名

sessionUserLogin.jsp

```

<%@ page contentType="text/html;charset=gb2312"%>
<script language="javascript">
    function on_submit(){//验证数据的合法性
        if (form1.username.value == ""){
            alert("用户名不能为空，请输入用户名！");
            form1.username.focus();
            return false;
        }
        if (form1.userpassword.value == ""){
            alert("用户密码不能为空，请输入密码！");
            form1.userpassword.focus();
            return false;
        }
    }
</script>
<%!
public String codeToString(String str){//处理中文字符串的函数
    String s=str;
    try{
        byte tempB[]=s.getBytes("ISO-8859-1");

```

```

        s=new String(tempB);
        return s;
    }catch(Exception e){
        return s;
    }
}
%>
<%
String username=request.getParameter("username");
String userpassword=request.getParameter("userpassword");
if(username!=null&userpassword!=null){
//如果用户名和密码都合法,则记下用户名,一般把用户和密码存在数据库中,
//用数据库中的信息与提交的用户名和密码比较以进行用户合法性检查,
//这些内容在后续章节中会继续学习
    session.setAttribute("username",codeToString(username));
    response.sendRedirect("sessionUserLogin1.jsp");
}
%>
<html>
<head>
<title>用户登录</title>
</head>
<body>
<table align="center">
<form name="form1" method="post" action="sessionUserLogin.jsp"
    onsubmit="return on_submit()">
<tr align="center">
<td>
用户登录
</td>
</tr>
<tr align="center">
<td>
请输入用户名: <input type="text" name="username" size="20">
</td>
</tr>
<tr align="center">
<td>
请输入密码: &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<input type="password" name="userpassword" size="20">
</td>
</tr>
<tr align="center">
<td>
<input type="submit" value="提交" name="B1">
<input type="reset" value="全部重写" name="B2">
</td>
</tr>
</form>
</table>
</body>
</html>

```

sessionUserLogin1.jsp

```
<%@ page contentType="text/html; charset=gb2312"%>
```



```

<html>
<head>
<title>用户登录成功</title>
</head>
<body>
用户登录成功! <br>
你的用户名是: <%= (String)session.getAttribute("username") %>
</table>
</body>
</html>

```

本示例把这两个文件放在同一个目录下。`sessionUserLogin.jsp` 文件中首先声明了一个表单，用于填写用户名和密码；文件的开始处是一段 JavaScript 程序，这段程序在客户端的浏览器中执行，用于验证用户提交的数据是否合法，这里仅验证两者不能为空；接下来是字符串的中文处理函数；如果数据合法，表单提交后，要判断用户名和密码是否为空。因为当表单没有提交时，程序片仍然会执行，用户名和密码一般被保存在数据库中，常用数据库中的信息与提交的用户名和密码进行比较，以进行用户合法性检查，这些内容在后续章节中会继续学习。检查用户名和密码则通过用一个 session 对象的数据对象 `username` 记住这个用户名，再将页面重定向至 `sessionUserLogin1.jsp` 页面。

在 `sessionUserLogin1.jsp` 中，用 `getAttribute()` 方法得到 session 对象的数据对象 `username` 的值并输出，由此发现利用 session 对象可以在同一个会话的不同网页中共享变量。

如图 5-8 所示为提交了表单后，重定向至 `sessionUserLogin1.jsp` 页面；`sessionUserLogin1.jsp` 页面的运行结果如图 5-9 所示。



图 5-8 sessionUserLogin.jsp 页面



图 5-9 sessionUserLogin1.jsp 页面

## 5.4.2 猜字母游戏

### 【实例 5-6】猜字母游戏

```

guessCharExample1.jsp

<%@ page contentType="text/html; charset=gb2312"%>
<html>
<head>
<title>猜字母游戏</title>
</head>
<body>
下面，我们一起来玩一个游戏：猜字母游戏<br>
游戏规则：电脑会随机自动生成一个字母，请你猜出这个字母是什么。字母忽略大小写。<br>

```

```

<% String charString=new String("abcdefghijklmnopqrstuvwxyz");
    int charNumber=((int)(Math.random()*100)+1)%26-1;
    Character TempCharacter=new Character(charString.charAt(charNumber));
    session.setAttribute("TempCharacter",TempCharacter);
    %>
<BR>
<P>输入你所猜的字母:
    <FORM action="guessResultExample1.jsp" method="post" name=form>
        <INPUT type="text" name="guesschar" >
        <INPUT TYPE="submit" value="提交" name="submit">
    </FORM>
</body>
</html>

```

#### guessResultExample1.jsp

```

<%@ page contentType="text/html;charset=gb2312"%>
<html>
<head>
<title>猜字母游戏</title>
</head>
<body>
<%
String tempString=request.getParameter("guesschar");
String TempCharacter=session.getAttribute("TempCharacter").toString();
if(tempString!=null){
    if(TempCharacter.equalsIgnoreCase(tempString))
        out.println("恭喜你,你猜对了!");
    else
        out.println("你猜错了,加油哦!");
}
%>
<BR>
<P>输入你所猜的字母:
    <FORM action="guessResultExample1.jsp" method="post" name=form>
        <INPUT type="text" name="guesschar" >
        <INPUT TYPE="submit" value="提交" name="submit">
    </FORM>
<a href="guessCharExample1.jsp">重新开始游戏</a>
</body>
</html>

```

本示例将这两个文件放在同一个目录中。

在 `guessCharExample1.jsp` 文件中,首先声明了一个包含 26 个英文字母的字符串数组;然后随机生成了一个 1~26 之间的整数数字,以这个整数数字为下标取对应字母的字符串中的字符,并将字符转换为一个字符串后放入 `session` 对象中;最后声明了一个表单,用于输入用户猜测的字母,并提交给 `guessResultExample1.jsp` 页面,如图 5-10 所示。

在 `guessResultExample1.jsp` 文件中,接收用户提交的猜测字母并与 `session` 对象中保存的字母进行比较,如果相等则报告给用户,表示猜对了,如果猜错了,则要求继续猜测;其中用到了字符串比较函数 `equalsIgnoreCase()`,这个方法在比较时忽略大小写,如图 5-11 所示。



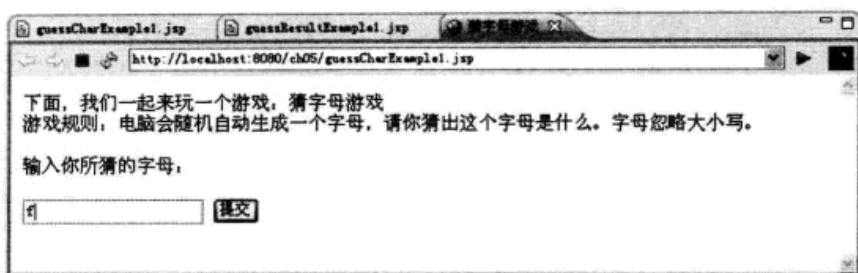


图 5-10 guessCharExample1.jsp 页面



图 5-11 guessResultExample1.jsp 页面

## 5.5 application 对象

application 对象用来在多个程序或者是多个用户之间共享数据，用户使用的所有 application 对象都是一样的，这与 session 对象不同。服务器一旦启动，就会自动创建 application 对象，并一直保持下去，直至服务器关闭，而 application 会自动消失。

### 5.5.1 application 对象的方法

application 对象实现了 javax.servlet.ServletContext 接口，此接口位于 servlet-api.jar 包中，代表当前 Web 应用上下文。application 对象的常用方法有如下一些。

#### 1. getAttribute()

此方法得到 Servlet 上下文属性 name 所代表的对象，如果没有找到则返回 null，调用方法如下：

```
public Object getAttribute(String name)
```

#### 2. getAttributeNames()

此方法得到 Servlet 上下文属性所有名称的枚举，调用方法如下：

```
public java.util.Enumeration getAttributeNames()
```

#### 3. getContext()

此方法得到指定的 URI 相应的 Servlet 上下文对象 ServletContext，参数 urlpath 为 URI，必

须是绝对路径，即以“/”开头，调用方法如下：

```
public ServletContext getContext(String urlpath)
```

#### 4. getInitParameter()

此方法返回一个 **String**，值为指定的当前 **Web** 应用上下文配置参数 **name** 的值，如果参数不存在则返回 **null**。**Web** 应用上下文配置参数可以在 **Web** 应用部署描述文件 **web.xml** 中进行定义，这个文件位于 **Web** 应用的 **WEB-INF** 子目录中，调用方法如下：

```
public String getInitParameter(String name)
```

#### 5. getInitParameterName()

此方法将当前 **Web** 应用上下文中的配置参数作为一个枚举返回，调用方法如下：

```
public java.util.Enumeration getInitParameterName()
```

#### 6. getRealPath()

此方法返回 **path** 所对应的全路径。如果无法解释为一个文件系统全路径（如 **path** 中的文件由 **WAR** 文件给出）则返回 **null**，调用方法如下：

```
public String getRealPath(String path)
```

#### 7. getServletContextName()

此方法得到当前 **Web** 应用描述文件中 **<display-name>** 元素所定义的上下文名称，调用方法如下：

```
public String getServletContextName()
```

#### 8. removeAttribute()

此方法删除 **Servlet** 上下文中指定的属性 **name** 所代表的对象，调用方法如下：

```
public void removeAttribute(String name)
```

#### 9. setAttribute()

此方法将 **Servlet** 上下文中指定的名称 **name** 所代表的对象设为 **attribute**，如果 **name** 已存在则替换为 **attribute**，如果不存在则创建一个，调用方法如下：

```
public void setAttribute(String name, Object attribute)
```

### 5.5.2 计数器

在第 4 章中，已编写过一个网站计数器的应用程序，但这个计数器在同一个客户端上每刷新一次网页时，计数器就会增 1，故计数并不准确。应当根据是否是一个新的会话来判断是否是一个新访问网站的用户，**application** 对象则可以用来保存访问的人数。下面是该计数器程序实例。



## 【实例 5-7】网站计数器

CounterApp1.jsp

```

<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="javax.servlet.*" %>
<HTML>
<head>
<title>网站计数器</title>
</head>
<BODY>
<%!
    synchronized void countPeople(){//串行化计数函数
        ServletContext application=((HttpServlet)(this)).getServletContext();
        Integer number=(Integer)application.getAttribute("Count");
        if(number==null){ //如果是第 1 个访问本站
            number=new Integer(1);
            application.setAttribute("Count",number);
        }else{
            number=new Integer(number.intValue()+1);
            application.setAttribute("Count",number);
        }
    }
%>
<% if(session.isNew())//如果是一个新的会话
    countPeople();
    Integer yourNumber=(Integer)application.getAttribute("Count");
%>
<P><P>欢迎访问本站，你是第
    <%=yourNumber%>
个访问用户。
</BODY>
</HTML>

```

程序用 `synchronize` 关键字对计数函数进行了串行化（有的书中叫序列化），以确保当两个客户端同时访问网页而修改计数值时不会产生冲突；`getServletContext()` 方法来得到 `application` 对象，因为有些 Web 服务器并不直接支持 `application` 对象，必须先得到其上下文；如果还是第一个访问的客户，则前面代码中得到的 `number` 会是空值，故置初值为 1，否则做增 1 处理；如果是一个新的会话则调用计数函数，得到计数值并将其显示。网站计数器程序的运行结果如图 5-12 所示。

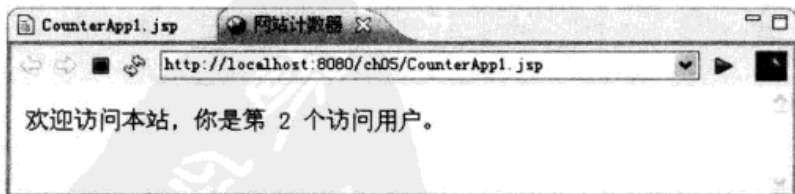


图 5-12 网站计数器的运行结果

可以发现，当刷新页面时，其数值并不会增加，只有关闭了本网站的所有窗口再重新访问

时，才会增 1，因为这又是一个新的会话。

## 5.6 out 对象

out 用来向客户端输出数据，其实在前面章节的实例中已经多次使用过，其最为常用的方法就是 print()及 println()方法。

### 5.6.1 out 对象的方法

out 对象代表向客户端浏览器输出内容的输出对象，由 Web 容器指定为 javax.servlet.jsp.JspWriter 类的一个子类，javax.servlet.jsp.JspWriter 类位于 jsp-api.jar 组件包中。javax.servlet.jsp.JspWriter 类的声明情况如下：

```
public abstract class javax.servlet.jsp.JspWriter extends java.io.Writer
```

可见 javax.servlet.jsp.JspWriter 类还继承了 java.io.Writer 类。

out 对象常用的方法有如下一些。

#### 1. clear()

此方法清空输出缓冲区中的内容，如果缓冲区已经刷新输出，则抛出一个 IOException 异常，以表示有些数据已被不可恢复地写至客户端，调用方法如下：

```
public void clear() throws java.io.IOException
```

#### 2. clearBuffer()

此方法清空输出缓冲区中的内容，与 clear()方法不同，如果缓冲区已经刷新输出，此方法并不抛出 IOException 异常，而是在清空缓冲区当前内容后返回，调用方法如下：

```
public void clearBuffer() throws java.io.IOException
```

#### 3. close()

此方法在刷新并输出 JspWriter 后将其关闭。如果在 close()调用之后再调用 flush()或 writer()则会抛出一个 IOException 异常，调用方法如下：

```
public void close() throws java.io.IOException
```

#### 4. flush()

此方法将输出缓冲区中的当前内容刷新输出，如果是有 Web 应用中则表示缓冲区中的内容会立即交付给客户端，调用方法如下：

```
public void flush() throws java.io.IOException
```

#### 5. getBufferSize()

此方法得到输出缓冲区的大小，单位为字节，如果无缓冲区则返回 0，调用方法如下：

```
public int getBufferSize()
```



## 6. isAutoFlush()

此方法是否设置为自动刷新输出缓冲区，是则返回 **true**，否则返回 **false**，调用方法如下：

```
public boolean isAutoFlush()
```

## 7. print()

此方法 **out** 对象的 **print()** 方法有许多种形式，主要用来在浏览器中输出数据。下面列出 **print()** 方法的各种形式，调用方法如下：

```
public void print(boolean arg0) throws java.io.IOException;
public void print(char arg0) throws java.io.IOException;
public void print(char[] arg0) throws java.io.IOException;
public void print(double arg0) throws java.io.IOException;
public void print(float arg0) throws java.io.IOException;
public void print(int arg0) throws java.io.IOException;
public void print(Object arg0) throws java.io.IOException;
public void print(String arg0) throws java.io.IOException;
public void print(long arg0) throws java.io.IOException;
```

以上方法的功能也就是将括号中的参数值转换为字符串后再输出。**print()** 方法是 **out** 对象最为常用的方法了，常用来输出字符串、输出 **HTML** 标签等内容，输出的内容由浏览器负责解释。

## 5.6.2 用 out 对象输出表格

### 【实例 5-8】用 out 对象输出表格

outApp1.jsp

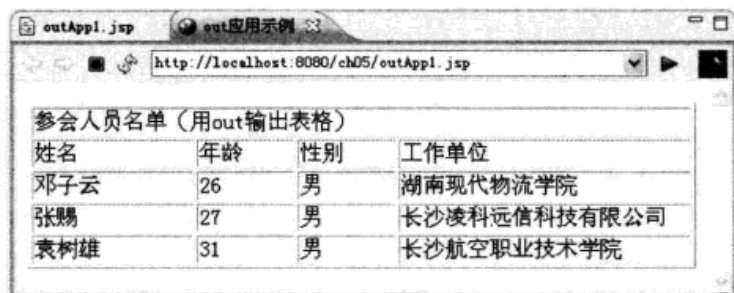
```
<%@ page contentType="text/html; charset=GB2312" %>
<HTML>
<head>
<title>out 应用示例</title>
</head>
<BODY>
<%
out.print("<table border='1' width='100%'><tr><td width='100%' colspan='4'>");
out.print("参会人员名单（用 out 输出表格）"+"</tr>");
out.print("<tr><td width='25%'>"+ "姓名"+"</td>");
out.print("<td width='15%'>"+ "年龄"+"</td>");
out.print("<td width='15%'>"+ "性别"+"</td>");
out.print("<td width='45%'>"+ "工作单位"+"</td></tr>");
out.print("<tr><td width='25%'>"+ "邓子云"+"</td>");
out.print("<td width='15%'>"+ "26"+"</td>");
out.print("<td width='15%'>"+ "男"+"</td>");
out.print("<td width='45%'>"+ "湖南现代物流学院"+"</td></tr>");
out.print("<tr><td width='25%'>"+ "张赐"+"</td>");
out.print("<td width='15%'>"+ "27"+"</td>");
out.print("<td width='15%'>"+ "男"+"</td>");
out.print("<td width='45%'>"+ "长沙凌科远信科技有限公司"+"</td></tr>");
out.print("<tr><td width='25%'>"+ "袁树雄"+"</td>");
out.print("<td width='15%'>"+ "31"+"</td>");
```

```

out.print("<td width='15%'>"+男+"</td>");
out.print("<td width='45%'>"+长沙航空职业技术学院+"</td></tr>");
out.print("</table>");
%>
</BODY>
</HTML>

```

该程序用 `out` 对象的 `print()` 方法输出了一系列的表格元素。在实际应用中，不仅是表格元素，任意 HTML 元素都可以用 `out` 对象输出，甚至是运行在客户端的脚本程序代码。如果程序与数据库进行了关联，就可以在 HTML 中的适当位置用 `out` 对象输出数据库中的数据信息，如程序示例中的姓名、年龄、性别及工作单位等数据信息。输出时要注意“”符号的配对使用，本程序的运行结果如图 5-13 所示。



姓名	年龄	性别	工作单位
邓子云	26	男	湖南现代物流学院
张赐	27	男	长沙凌科远信科技有限公司
袁树雄	31	男	长沙航空职业技术学院

图 5-13 用 `out` 对象输出表格

JSP 中还有一些其他的内置对象，如 `config`。`config` 对象实质上是 `ServletConfig` 类的一个对象，用来配置处理 JSP 页面的句柄，仅在页面范围内有效；这些对象在 JSP 中很少被用到，故在此不再详述。

## 5.7 小结

在本章中，讲述了 JSP 内置对象的应用，重点讲述了 `request`、`response`、`session`、`application`、`out` 这五个常用的对象。

`request` 对象封装了客户端提交的数据信息以及客户端的一些信息，如客户端用表单提交的数据；`response` 对象用来对客户请求做出响应，向客户端发送数据；`session` 对象用来保存一些需要在与每个用户会话期间保持的数据信息；`application` 对象用来在多个程序或者是多个用户之间共享数据；`out` 用来向客户端输出数据。

## 5.8 练习

1. 修改 5.5 节中的计数器，结合 `application` 与 `session` 这两个对象编写一个计数器，当一个新的用户会话开始时才计数。（程序代码可以参考实例 5-7）
2. 编写一个用户注册网页，用 JavaScript 检验数据的合法性；用 `request` 对象接收提交的数据，并显示出提交的数据（程序代码可以参考实例 5-2）。



# 06

## JSP 中数据库的使用

在计算机信息管理系统中，管理各种资料最好的办法就是使用数据库。数据库是一个专门用来存放和管理各种数据的地方，目前比较成熟的是关系型数据库。在应用程序中需要访问这些数据库，以不断获得和更新数据，JSP 也不例外。在 JSP 中主要使用 JDBC（Java Database Connectivity，Java 数据库连接）技术来连接数据库，它有四种类型的驱动，包括 JDBC-ODBC Bridge、JDBC Native Bridge、JDBC Net Bridge、Pure Java JDBC Driver。另外，连接池技术扩展了 JDBC 的应用能力，使得大量数据库的连接得以快速实现。

通过本章的学习，读者应当能够利用 JDBC 来处理在 JSP 中各种操作数据库的应用，配置数据库连接池。掌握好 JSP 中数据库的使用是开发一个基于 Java 的 B/S 信息系统的关键技术，也是本书的重点。

### 6.1 SQL 基础

SQL（Structured Query Language，结构化查询语言）是操作数据库的一种高级语言，数据库管理员、程序员都通过它来操作数据库，虽然现在一些图形化的管理工具使得管理工作越来越容易，管理者不必输入 SQL 语句，但这些工具大多是把图形化的操作转化为 SQL 语句再由数据库管理系统执行。在应用程序的程序设计中需要程序员用 SQL 语句来访问数据库中的数据，因此掌握 SQL 语言的基本知识是很有必要的。如果读者对 SQL 语言已经比较了解可以跳过 6.1 节的内容，直接学习 6.2 节。

数据库管理系统的发展经历四个发展阶段：层次型、网状型、关系型及关系对象型。目前最为成熟的是关系型，现在大多数的数据应用系统使用的数据库都是关系型数据库，应用较为广泛的数据库有：Oracle、Sybase、Informix、Microsoft SQL Server、DB、MySQL、FoxPro、Microsoft Access 等。

Microsoft SQL Server 是 Microsoft 公司的大型关系型数据库产品，简单易用，用户界面友好，可伸缩性较好，适用于大、中、小各种级别的应用场合。本书中的数据库操作实例均是基于

Microsoft SQL Server 来实现的，读者的机器如果配置不高、资源有限的话，可使用轻量级的数据库系统——Microsoft Access，它是 office 套件的一部分，在安装 office 时就可以安装。Access 占用的系统资源较少，用来学习和开发小型的信息系统，同时它也支持 SQL 语言、存储过程等功能。那么，下面就一起来学习 SQL 语言的基本知识。

### 6.1.1 表操作

SQL 语句是一种标准化的结构化查询语言，各种不同关系型数据库的 SQL 语句大同小异。在 SQL Server 中 SQL 语句对大小写不敏感。下面先来学习关于表操作的相关语句。

#### 1. 新建表

Create Table 语句用来建立数据库中的表。表就像日常工作中的二维表格，用来存放数据。建表的基本语法如下。

```
Create Table tablename(column_name1 datatype [column_constraint],
                        column_name 2 datatype [column_constraint],
                        ...)
```

其中，tablename 是要建立表的名称；column\_name1 是建立表中列的名称，即字段名；datatype 是这个字段的数据类型；column\_constraint 是字段的约束规则，如：主键约束、外键约束等。

一般在 JSP 应用程序中不会有诸如创建数据库、删除数据库这样的操作，建立数据库后，可在应用程序中操作数据，完成数据的插入、删除和更新操作。在 SQL Server 中建立数据库的操作过程如下（以 SQL Server 2008 为例）。

首先打开“SQL Server Management Studio”，会弹出“连接到服务器”对话框，在其中输入连接参数，比如“服务器名称”、“身份验证”、“登录名”、“密码”。连接上数据库后，在树形目录中的“数据库”上单击右键，选择“新建数据库(N)...”，会弹出“新建数据库”对话框，输入数据库的名称，单击“确定”按钮即可建立数据库。在 SQL Server 中，数据库的大小会自动进行调整，一般场合下的应用不必配置过多的参数。建立数据表可直接在企业管理器中以图形化的方式操作。

如果要建立一个数据表 userTable 以保存网站注册用户的信息，假设用前面的方法建立了一个数据库 testDatabase，那么就可用 SQL 语句建立表。SQL 语句如下：

```
CREATE TABLE userTable (
    user_id bigint IDENTITY (1,1) NOT NULL primary key,
    user_name varchar(40),
    user_password varchar(40),
    user_true_name varchar(40),
    user_age int,
    user_sex varchar(2),
    user_address varchar(80),
    user_telephone varchar(20),
    add_time datetime DEFAULT (getdate())
)
```

输入完语句后，按工具栏的“执行查询”按钮，即可执行 SQL 语句。查询框的下面是消息



框，可见命令已成功执行。

SQL 语句中的“**IDENTITY (1,1)**”表示自动增量，每次自动增 1，其值由系统自动生成；**user\_id** 表示用户 ID 表，为主键，故不能为空；其他字段如 **user\_name** 表示用户名，**user\_password** 表示用户密码，这是登录时要使用的，**user\_true\_name** 表示用户的真实姓名，**user\_age** 表示用户年龄，**user\_sex** 表示用户性别，**user\_address** 表示用户住址，**user\_telephone** 表示用户电话号码，**add\_time** 表示添加用户的时间，默认值为加入记录时的系统当前时间。至此建表操作已完成。

**bigint**、**varchar**、**int**、**datetime** 这些是 SQL Server 中的数据类型，可根据需要进行设置。**bigint** 表示整型，它比 **int** 表示的范围要大；**varchar** 表示变长字符串型，**datetime** 为日期型。

## 2. 删除表

删除表用 **drop table** 语句，语法格式如下。

```
drop table table_name
```

如前面建立的 **userTable** 表，可用如下的语句删除：

```
drop table userTable
```

## 3. 更改表

更改表用 **alter table** 语句，可用来更改、添加、除去列和约束，其语法格式如下。

```
alter table table_name
[alter column column_name new_datatype [column_constraint]]
|[add column_name new_datatype [column_constraint]]
|[drop column_name new_datatype [column_constraint]]
```

如要把 **userTable** 表中的用户名 **varchar** 类型长度改为 20，则可用如下的 SQL 语句。

```
alter TABLE userTable
alter column user_name varchar(20)
```



**注意** 更改表字段类型时，数据有可能会有精度受损；一旦确定字段名就不能更改，如果要更改字段名，须删除原来的字段，再新增字段。

### 6.1.2 查询语句

在 SQL 语句中，查询语句 **select** 是最为常用的，其基本的语法格式如下。

```
SELECT select_list [ INTO new_table ]
FROM table_source
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

其中，**select\_list** 是要查询的内容，如：字符名称的列表；**[ INTO new\_table ]**是指可以把查询的结果放入一个新的表中；**table\_source** 是指表名，是查询数据的来源，如果是单表查询则为一个表的名称，如为多表查询则是表的名称列表；**search\_condition** 是查找的条件；**group\_by\_expression** 是查询结果分组的条件；**search\_condition** 中分组后组内的条件；**order\_expression**

[ASC | DESC]是指排序的表达式以及方式，如：`order_expression` 可以是一个字段名，则根据这个字段来排序，ASC 表示升序，DESC 表示降序。

如果在 `userTable` 中要找出年龄在 20~30 岁之间的用户信息，并按 ID 号升序排序，相应的 SQL 语句如下。

```
select * from userTable
  where user_age>10 and user_age<30
 order by user_id asc
```

其中 “\*” 表示查询出 `userTable` 表中的所有字段。

在 `select` 语句中，`where` 子句可以用来限制查询记录的条件，在上面的 SQL 查询语句中已经使用过了。对于 `where` 子句作出以下说明。

### 1. =

如果数据类型是数字型则不必用引号（'或"）括起来，如果是字符型，则必须用引号括起来。如：

```
select * from userTable
  where user_age=20 and user_name='yourname'
```

其功能是从 `userTable` 表中查找出年龄是 20，姓名是 `yourname` 的记录。

### 2. in 与 not in

`in` 用来确定给定的值是否与子查询或列表中的值相匹配，使用 `not in` 则对返回值取反。其语法格式如下：

```
test_expression [ NOT ] IN
(
  subquery
  | expression [ , ...n ]
)
```

其中，`test_expression` 是一个有效的表达式；`subquery` 是包含某列结果集的子查询，必须与 `test_expression` 具有相同的类型；`expression [,...n]` 是一个表达式列表，用来测试是否匹配，所有的表达式必须和 `test_expression` 具有相同的类型。

如：查询出年龄是 20、22、25 的用户信息。

```
select * from userTable
  where user_age in (20,22,25)
```

### 3. between...and 与 not between...and

`between...and` 用来选择列值在某个范围内的记录，`not between...and` 则恰恰相反，用来选择列值不在某个范围内的记录，其语法如下。

```
test_expression [ NOT ] BETWEEN begin_expression AND end_expression
```

如：查询出年龄在 20~25（包括 20 岁和 25 岁）岁之间的用户信息。

```
select * from userTable
  where user_age between 19 and 26
```



#### 4. like 与 not like

确定给定的字符串是否与指定的模式匹配。模式可以包含常规字符和通配符字符，可使用字符串的任意片段匹配通配符。与使用“=”和“!=”字符串比较运算符相比，使用通配符可使 LIKE 运算符更加灵活。其语法格式如下：

```
match_expression [ NOT ] LIKE pattern
```

其中，`match_expression` 是任何字符串数据类型的有效表达式，`pattern` 是在 `match_expression` 中的搜索模式，可有如下的通配符：“%”表示包含零个或多个字符的任意字符串；“\_”（下画线）表示任意单个字符。通配符在各种数据库中可能会不一样，这里仅介绍 SQL Server 的通配符。

如：查询出真实姓名为姓“邓”的用户信息。

```
select * from userTable
where user_true_name like '邓%'
```

#### 5. is null 与 is not null

`is null` 确定一个给定的表达式是否为 NULL，`is not null` 与 `is null` 相反。其语法格式如下：

```
expression IS [ NOT ] NULL
```

其中 `expression` 是一个有效的表达式。

如：查询出真实姓名是空的用户信息。

```
select * from userTable
where user_true_name is null
```

#### 6. and 与 or

`and` 运算用来连接两个布尔型表达式并当两个表达式都为 `true` 时返回 `false`。

`or` 运算用来将两个条件结合起来，满足其中之一即返回 `true`。

`and` 运算的优先级比 `or` 要高。

### 6.1.3 插入、更新与删除语句

#### 1. 插入

插入时使用 `insert` 语句，语法如下。

```
insert into table_name(column_name1, column_name2, ...)
values(column_name1_values, column_name2_values, ...)
```

其中，`table_name` 是将行添加到数据库表中；`(column_name1, column_name2, ...)` 是要添加内容的列；`(column_name1_values, column_name2_values, ...)` 是将要插入到单个列中对应的值或表达式。

如：要往用户表中插入一条用户信息记录。

```
insert into userTable(user_name,user_password,user_true_name,
user_age,user_sex,user_address,user_telephone)
values('dzy','6582254','邓子云',26,'男','长沙市商业银行信息技术部',
'0731-6582254')
```



**注意** 这个表的 `user_id` 与 `add_time` 字段的值可以自动生成，不必给出。

## 2. 更新

修改数据库表中记录的数据用 `update` 语句，其语法格式如下。

```
update table_name set column_name1= column_name1_values, column_name2=
column_name1_values,...[where 子句]
```

其中，`where` 子句同 `select` 语句中的 `where` 子句。

如：把用户表中真实姓名姓邓的年龄全部改为 27 岁。

```
update userTable set user_age=27 where user_true_name like '邓%'
```

## 3. 删除

删除数据库表中的记录时用 `delete` 语句，语法格式如下。

```
delete from table_name [where 子句]
```

`where` 子句同 `select` 语句中的 `where` 子句。

如：删除用户表中姓邓的用户的所有记录。

```
delete from userTable where user_true_name like '邓%'
```

## 6.1.4 存储过程

存储过程是指存储在数据库管理系统中，由一段 SQL 语句组成的一个过程。使用存储过程可以加快程序的执行速度，因为储存的 SQL 语句可以预编译，可以保证事务的原子性。执行一个存储过程，可以在其中作出事务处理，这样可以保证存储过程中的多条 SQL 语句一起全部执行，或者都不执行。适用于需要应用事务的场合，如：银行中记账是复式记法，既有账户加钱，又有账户减钱，要保证两个账户的这两个操作都进行，或者都不进行，可以把这两个数据库的更新操作作为一个事务可以把多条 SQL 语句放在一个存储过程中。

在 SQL Server 中建立存储过程的语法格式如下。

```
create procedure procedure_name[@parameter data_type][=default][OUTPUT]
    [ , ...n ]
as
    sql_statement [ ...n ]
```

其中，`procedure_name` 是新建存储过程的名称；`@parameter` 是过程中的参数。在 `CREATE PROCEDURE` 语句中可以声明一个或多个参数，用户必须在执行过程时提供每个声明参数的值（除非定义了该参数的默认值），使用 `@` 符号作为第一个字符来指定参数名称；`data_type` 是参数的数据类型；`default` 是参数的默认值；`OUTPUT` 表明参数是返回参数，该选项的值可以返回给 `EXEC[UTE]`，使用 `OUTPUT` 参数可将信息返回给调用过程，`sql_statement` 是过程中要包含的任意数目和类型的 SQL 语句。



执行存储过程的方法是：

```
EXEC [ UTE ] procedure_name parameter_values[ , ...n ]
```

如：下面的存过程把用户表中真实姓名中姓邓的用户的年龄改为指定的岁数，并返回用户表中所有记录。

```
create procedure update_age @age int
as
update userTable set user_age=@age
where user_true_name like '邓%'
select * from userTable
go
```

调用方法为（如改为 31 岁）：

```
execute update_age 31
```

## 6.2 JDBC

JDBC 定义了 Java 与各种 SQL 数据库之间的编程接口，JDBC API 是一个统一的标准应用程序编程接口，这样可以屏蔽异种数据库之间的差异。

### 6.2.1 JDBC 工作原理

JDBC 与 ODBC（OpenData Base Connectivity，开放式数据库连接）的作用非常类似，它在应用程序和数据库之间起到桥梁作用。ODBC 使用得特别广泛，在 Java 中还提供了 JDBC-ODBC 桥，能够在 JDBC 与 ODBC 之间进行转换，这样可以通过 ODBC 屏蔽不同种数据库之间的差异。

在 JDK 的包 `java.sql.*` 中定义的一系列类（Class）、接口（Interface）、例外（Exception）以及这些类和接口中定义的属性（property）和方法（method），JSP 的开发人员通过一定的规则调用 `java.sql.*` 中的 API 就可以实现对数据库管理系统的访问。JDBC 提供了接口，但具体类的实现则要求数据库的设计者完成。通过生成这些接口的实例，即使对于不同的数据库，Java 程序也可以正确地执行 SQL 调用。

### 6.2.2 JDBC 的四种驱动

#### 1. JDBC-ODBC Bridge

用 JDBC-ODBC Bridge 可以访问一个 ODBC 数据源，但在执行 Java 程序的机器上必须安装 ODBC 驱动，并作出配置。在 6.2.3 节中会告诉读者如何配置一个 ODBC 数据源。它的调用方式如图 6-1 所示。



图 6-1 JDBC-ODBC Bridge 调用

从上图可以看出，中间存在一个 JDBC-ODBC 的转换过程，影响了执行的效率。

## 2. JDBC Native Bridge

这种方式需要在 JSP 程序执行的机器上安装本地的且针对特定数据库的驱动程序，通过这个程序把对数据库的 JDBC 调用转换为数据库的 API 调用，因此其性能比 JDBC-ODBC 的方式要更好一些，缺点就是需要安装驱动程序。它的调用方式如图 6-2 所示。

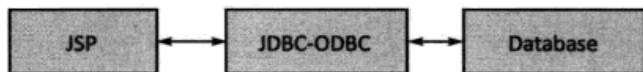


图 6-2 JDBC-Native 调用

## 3. JDBC-Network Bridge

这种方式不需要安装驱动程序，直接用 JDBC 通过网络连接数据库，因此与平台无关，效率较高，所以在 Internet 的 Web 开发中被大量应用。它的调用方式如图 6-3 所示。

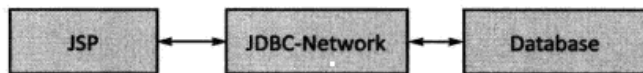


图 6-3 JDBC-Network 调用

## 4. Pure Java JDBC Drive

Java 驱动程序运行在客户端上，因此客户端可以直接访问数据库，其体系结构特别简单，但相应的安全性及程序的逻辑性不好，其调用方式如图 6-4 所示。

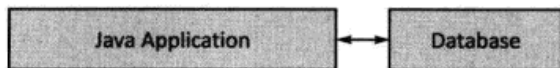


图 6-4 Pure Java JDBC 调用

### 6.2.3 ODBC 数据源

ODBC 数据源可以针对不同的数据库，在 Windows 操作系统中已经集成了一些数据库的 ODBC 驱动，如 Access、Excel 等，如果没有所需的 ODBC 驱动，则需要安装它，一般的数据库厂商会在其 Internet 网站上免费提供。

下面介绍怎样建立一个连接 SQL Server 的 ODBC 数据源。建立 ODBC 数据源的步骤如下。

① 打开“控制面板”→“系统和安全”→“管理工具”，双击“数据源（ODBC）”图标，出现如图 6-5 所示的窗口，选择“系统 DSN”选项卡。



**注意** 用户 DSN 即用户数据源，只对当前用户可见，且只能用于本机器；系统 DSN 即系统数据源对本机器上所有用户可见；文件 DSN 可由安装了相同驱动的不同机器上的用户共享。

对于系统 DSN，数据源的配置信息保存在 Windows 的注册表里，程序就是通过相应的注册



表信息来确定数据源的。

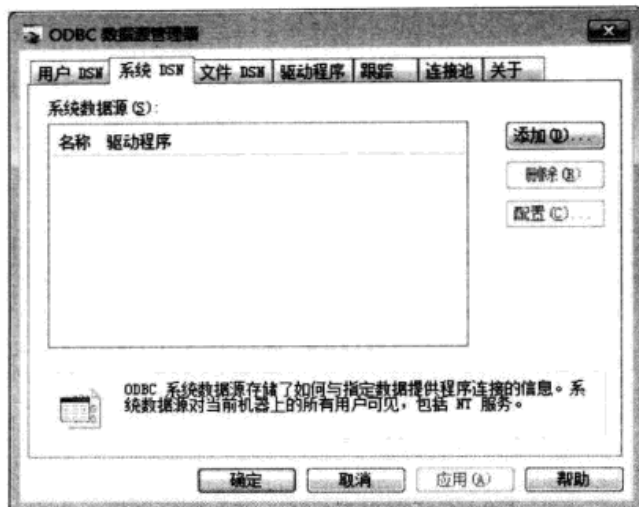


图 6-5 ODBC 数据源管理器

② 单击图 6-5 右边的“添加”按钮，弹出“创建新数据源”对话框，如图 6-6 所示，在这里选择一个正确的驱动程序，选择“SQL Server”，单击“完成”按钮。

③ 系统弹出“建立到 SQL Server 的新数据源”对话框，如图 6-7 所示，输入数据源的名称，名称可以自由设定，只要不与其他系统 DSN 同名即可。描述数据源的内容可以不写，写下一些文字来描述是为了方便今后的使用。给出服务器名称，如果是本机用“localhost”或 IP 地址“127.0.0.1”，或者本机真实 IP 地址或本机机器名都可以，如果是别的计算机可以用机器名或 IP 地址。单击“下一步”按钮，弹出如图 6-8 所示的对话框。

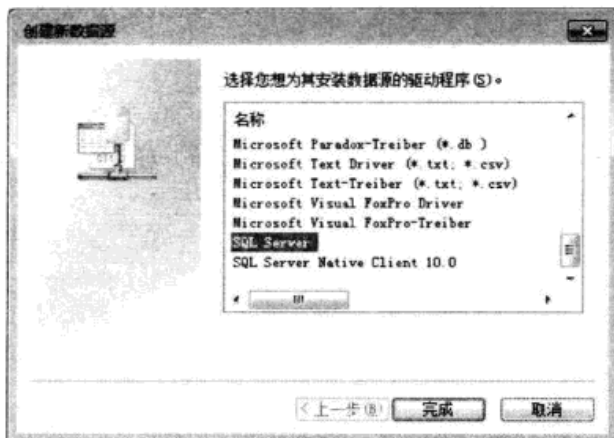


图 6-6 创建新数据源



图 6-7 建立新的数据源到 SQL Server

④ 选择“使用用户输入登录的 ID 和密码的 SQL Server 验证”，输入登录的 ID 和密码，这里使用了 SQL Server 最大权限的账户“sa”，实际应用中数据库会有一些权限控制，可能不会使用这个用户。单击“下一步”按钮，弹出如图 6-9 所示的对话框。

⑤ 选中“更改默认的数据库为”复选框，在下拉框中选择要连接的数据库，如本例中要连接用户注册表的数据库。单击“下一步”按钮，在接下来的对话框中单击“完成”按钮，再

在接下来的对话框中单击“确定”按钮，即完成了整个数据源的建立过程。在最后一个对话框中，有一个“测试数据源”的按钮可以测试数据源的配置是否正确，如果正确会显示“连接成功”的对话框，如是不正确会报错，可根据情况更改相应的配置。成功后会在第一步中的“ODBC 数据源管理器”的系统 DSN 中出现新建立的数据源，如本例中的“testDatabase”。

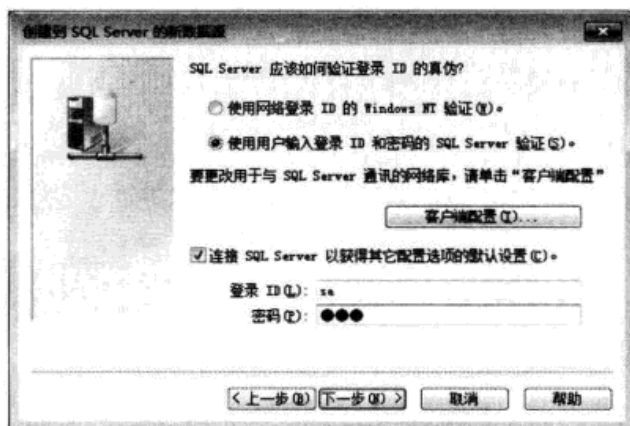


图 6-8 建立数据源的配置

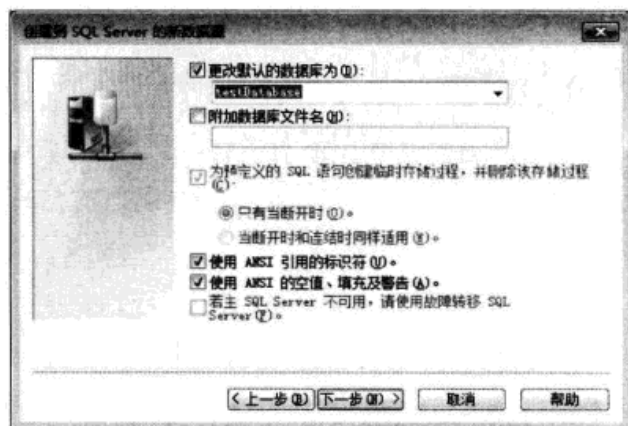


图 6-9 建立数据源的配置

建立其他数据库的 ODBC 数据源会稍有不同，如果是 Access 数据库，在第 2 步中选择“Microsoft Access Driver(\*.mdb)”驱动，下一步就和 SQL Server 不同了，因为 Access 是文件型数据库，选择相应的数据文件即可，整个过程将相对简单一些。

### 6.2.4 SQL Server 的 JDBC 安装

安装 SQL Server 的 JDBC 驱动方法，不同的 SQL Server 版本有所不同。为简便起见，可直接下载 Microsoft SQL Server JDBC Driver 3.0 或 4.0。这是一个 Type 4 JDBC 驱动程序，它通过 Java Platform, Enterprise Edition 5 中可用的标准 JDBC 应用程序编程接口 (API) 提供数据库连接。此版本的 JDBC 驱动程序与 JDBC 4.0 兼容，并在 Java 开发工具包 (JDK) 5.0 版本或更高版本上运行。

可以从 Microsoft 的下载中心找到 Microsoft SQL Server JDBC Driver 3.0 或 4.0 并下载。下载后得到的实际上是一个压缩包，解压缩后可以得到 2 个文件：sqljdbc4.jar 和 sqljdbc.jar。其中 sqljdbc.jar 类库提供对 JDBC 3.0 的支持，sqljdbc4.jar 类库提供对 JDBC 4.0 的支持，它不仅包括 sqljdbc.jar 的所有功能，还包括新增的 JDBC 4.0 方法，可见我们使用 sqljdbc4.jar 就可以了。

如果只需要在当前 Web 应用中可用，则可将 sqljdbc4.jar 拷贝到 Web 应用的“WEB-INF/lib”目录下。如果要在所有的应用中都可用，则拷贝到 Tomcat 7 的 lib 目录下。

在 JDBC API 4.0 中，DriverManager.getConnection 方法得到了增强，可自动加载 JDBC Driver。因此，使用 sqljdbc4.jar 类库时，应用程序无须调用 Class.forName 方法来注册或加载驱动程序。

调用 DriverManager 类的 getConnection 方法时，会从已注册的 JDBC Driver 集中找到相应的驱动程序。sqljdbc4.jar 文件包括“META-INF/services/java.sql.Driver”文件，后者包含 com.microsoft.sqlserver.jdbc.SQLServerDriver 作为已注册的驱动程序。现有的应用程序（当前通



过使用 `Class.forName` 方法加载驱动程序)将继续工作,而无须修改。



**注意** `sqljdbc4.jar` 类库要求使用 6.0 或更高版本的 Java 运行时环境 (JRE)。

## 6.2.5 JDBC 接口

### 1. Driver

建立了 ODBC 数据源后,就可以在 Java 程序中通过 JDBC-ODBC 接口连接数据库了。无论是用 JDBC-ODBC 还是直接用 JDBC,在用之前,都要在文件前导入有关 SQL 的类,可用下面的语句。

```
import java.sql.*;
```

使用 JDK 7,应用程序不必再通过 `Class.forName()`方法来加载数据库驱动了。取而代之的是, `DriverManager` 会根据应用请求连接的情况,自动查找到合适的 JDBC 驱动。

### 2. DriverManager

用 `DriverManager` 的 `getConnection()`方法可以创建一个数据库连接对象,它的一般形式如下。

```
public static Connection getConnection(String url) throws SQLException;
public static Connection getConnection(String url, String user, String password)
    throws SQLException;
public static Connection getConnection(String url, java.util.Properties info)
    throws SQLException;
```

参数 `user` 是指连接数据库时使用的用户名;参数 `password` 是指连接数据库时的用户 `user` 的登录密码;参数 `info` 是作为连接参数的任意字符串标记/值对的列表,通常至少应该包括“`user`”和“`password`”属性;参数 `url` 是指连接数据库的 URL。该方法返回一个连接,一般用一个连接对象接收返回对象。

当驱动类型是 JDBC-ODBC 时用 `jdbc:odbc`,如果是 JDBC 时就用 `JDBC`。其余参数不同的数据库有些不同,如下所示。

```
//SQL Server
DriverManager.getConnection("jdbc:sqlserver://主机\实例名:端口号;
    DatabaseName=数据库名","用户名","密码")
```

如要连接本机的 `testDatabase` 数据库,用户名为 `sa`,密码为 `123`,则用如下的语句。

```
Connection conn=DriverManager.getConnection
    ("jdbc:sqlserver://localhost:1433;
    DatabaseName=testDatabase","sa","123");
```

连接 SQL Server 的常用端口号是 1433。连接其他数据库的连接字符串如下。

```
//MySQL
DriverManager.getConnection("jdbc:mysql://主机:端口号:数据库名","用户名","密码")
//Oracle
DriverManager.getConnection("jdbc:Oracle:thin:@主机:端口号:数据库名","用户名","密码")
//Informix
DriverManager.getConnection("jdbc:informix-sqli://主机:端口号/数据库名:
    INFORMIXSERVER=informix 服务名","用户名","密码")
```

```
//Sybase
DriverManager.getConnection("jdbc:sybase:Tds:主机:端口号/数据库名","用户名","密码")
//AS400
DriverManager.getConnection("jdbc:as400://主机","用户名","密码")
```



**注意** 请读者注意区分不同的数据库连接中的细微差别，它们的符号和字符稍有差异。

连接数据库有时会不成功，在程序中应及时捕获异常，以增强程序的健壮性，如连接本机，则用如下的语句。

```
try{
    Connection conn=
        DriverManager.("jdbc:sqlserver://localhost:1433;
            DatabaseName=testDatabase","sa","123");
}catch(SQLException e){}
```

### 3. Connection

一个 **Connection** 对象表示与特定数据库的连接，主要用于执行 SQL 语句并得到执行的结果。默认情况下，**Connection** 对象处于自动提交模式下，也就是说它在执行每个 SQL 语句后都会自动提交更改，如果禁用自动提交模式，为了提交更改就必须显示调用 **commit()** 方法，否则将无法保存数据库更改。

**Connection** 有 5 个常量属性用于表示 JDBC 数据操作的事务级别：**TRANSACTION\_NONE**、**TRANSACTION\_READ\_COMMITTED**、**TRANSACTION\_READ\_UNCOMMITTED**、**TRANSACTION\_REPEATABLE\_READ**、**TRANSACTION\_SERIALIZABLE**，具体情况如表 6-1 所示。

表 6-1 JDBC 数据操作的事务级别

事务级别	值	脏 读	不可重复读	虚 读
<b>TRANSACTION_NONE</b>	0	可能发生	可能发生	可能发生
<b>TRANSACTION_READ_UNCOMMITTED</b>	1	可能发生	可能发生	可能发生
<b>TRANSACTION_READ_COMMITTED</b>	2	不可能发生	可能发生	可能发生
<b>TRANSACTION_REPEATABLE_READ</b>	4	不可能发生	不可能发生	可能发生
<b>TRANSACTION_SERIALIZABLE</b>	8	不可能发生	不可能发生	不可能发生

**TRANSACTION\_NONE** 表示不支持事务处理。

**TRANSACTION\_READ\_COMMITTED** 级别下，可以发生脏读、不可重复读和虚读，此级别允许由某一事务更改的行在已提交该行中的所有更改之前被另一个事务读取（“脏读”），如果所有更改都被回滚，则第二个事务将检索无效的行。



**注意** 数据库中行与记录、列与字段的概念，在本书中并不作严格区分，行与记录均是指数据库表中的一行数据或 JDBC 结果记录集中的一行数据；列与字段均是指数据库表中的一列数据或 JDBC 结果记录集中的一列数据。



TRANSACTION\_READ\_UNCOMMITTED 级别下, 能防止发生脏读, 但不可重复读和虚读有可能发生, 此级别只禁止事务读取其中带有未提交更改的行。

TRANSACTION\_REPEATABLE\_READ 级别下, 能防止发生脏读和不可重复读, 但可能发生虚读, 此级别禁止事务读取其中带有未提交更改的行, 它还禁止这种情况: 一个事务读取某一行, 而另一个事务更改该行, 第一个事务重新读取该行, 并在第二次读取时获得不同的值 (“不可重复读” )。

TRANSACTION\_SERIALIZABLE 级别下, 能防止发生脏读、不可重复读和虚读, 此级别包括 TRANSACTION\_REPEATABLE\_READ 中禁止的事项并进一步禁止出现这种情况: 某一事务读取所有满足 WHERE 条件的行, 另一个事务插入一个满足 WHERE 条件的行, 第一个事务重新读取满足相同条件的行, 并在第二次读取时检索到额外的 “虚” 行。

Connection 常用的方法有如下一些。

#### (1) close()

此方法用来立即释放此 Connection 对象的数据库和 JDBC 资源, 而不是等待它们被自动释放。在已经关闭的 Connection 对象上调用 close() 方法将什么事也不做, 调用方法如下:

```
void close() throws SQLException
```

#### (2) commit()

此方法使自从上一次提交/回滚以来进行的所有更改成为持久更改, 并释放此 Connection 对象当前保存的所有数据库锁定。此方法应该只在已禁用自动提交模式时使用, 调用方法如下:

```
void commit() throws SQLException
```

#### (3) createStatement()

此方法创建一个 Statement 对象将 SQL 语句发送到数据库。没有参数的 SQL 语句通常使用 Statement 对象执行。如果多次执行相同的 SQL 语句, 使用 PreparedStatement 对象可能更有效。使用返回的 Statement 对象创建的结果记录集在默认情况下类型为 TYPE\_FORWARD\_ONLY, 并带有 CONCUR\_READ\_ONLY 并发级别。调用方法如下:

```
Statement createStatement() throws SQLException
```

createStatement() 方法还有两种带有参数的形式:

```
Statement createStatement(int resultSetType, int resultSetConcurrency)
    throws SQLException;
```

```
Statement createStatement(int resultSetType, int resultSetConcurrency,
    int resultSetHoldability) throws SQLException;
```

第一种形式用于创建一个 Statement 对象, 该对象将生成具有给定类型和并发性的 ResultSet 对象, 此方法与不带参数的 createStatement() 方法功能类似, 但它允许重写默认结果记录集类型, 可以设置并发性。

第二种形式用于创建一个 Statement 对象, 该对象将生成具有给定类型、并发性和可保存性的 ResultSet 对象, 此方法与不带参数的 createStatement() 方法功能类似, 但它允许重写默认结果记录集类型, 可以设置并发性和可保存性。

resultSetType 参数的值可以取以下 ResultSet 常量之一: ResultSet.TYPE\_FORWARD\_ONLY、ResultSet.TYPE\_SCROLL\_INSENSITIVE 或 ResultSet.TYPE\_SCROLL\_SENSITIVE。

`resultSetConcurrency` 参数可以取以下 `ResultSet` 常量之一：`ResultSet.CONCUR_READ_ONLY` 或 `ResultSet.CONCUR_UPDATABLE`。

`resultSetHoldability` 参数可以取以下 `ResultSet` 常量之一：`ResultSet.HOLD_CURSORS_OVER_COMMIT` 或 `ResultSet.CLOSE_CURSORS_AT_COMMIT`。



**提示** `resultSetType` 参数、`resultSetConcurrency` 参数和 `resultSetHoldability` 参数的具体取值含义将在 `ResultSet` 接口中作详细说明。

以上两种形式的 `createStatement()` 方法如果发生数据库访问错误，或者给定参数不是指定类型、并发性和可保存性的 `ResultSet` 常量都将抛出 `SQLException` 异常。

#### (4) `getAutoCommit()`

检索当前 `Connection` 对象的自动提交模式。如果是 `true` 表示是自动提交模式，如果是 `false` 表示禁用自动提交模式，而需要开发人员在程序中显式地使用 `commit()` 方法提交事务。调用方法如下：

```
boolean getAutoCommit() throws SQLException
```

#### (5) `getMetaData()`

获取 `DatabaseMetaData` 对象，该对象包含关于 `Connection` 对象连接到的数据库的元数据。元数据包括关于数据库的表、受支持的 SQL 语法、存储过程、此连接的功能等信息。调用方法如下：

```
DatabaseMetaData getMetaData() throws SQLException
```

#### (6) `getTransactionIsolation()`

得到当前 `Connection` 对象的当前事务隔离级别。事务隔离级别的值是以下常量之一：`Connection.TRANSACTION_READ_UNCOMMITTED`、`Connection.TRANSACTION_READ_COMMITTED`、`Connection.TRANSACTION_REPEATABLE_READ`、`Connection.TRANSACTION_SERIALIZABLE` 或 `Connection.TRANSACTION_NONE`。调用方法如下：

```
int getTransactionIsolation() throws SQLException
```

#### (7) `isClosed()`

检查当前 `Connection` 对象是否已经被关闭。如果已经连接上，在调用 `close()` 方法或者发生某些严重的错误后会关闭 `Connection`。`isClosed()` 方法只保证在已经调用 `Connection.close()` 方法之后被调用时返回 `true`。调用方法如下：

```
boolean isClosed() throws SQLException
```

#### (8) `isReadOnly()`

判断当前 `Connection` 对象是否处于只读模式，是返回 `true`，否则返回 `false`。调用方法如下：

```
boolean isReadOnly() throws SQLException
```

#### (9) `prepareCall()`

创建一个 `CallableStatement` 对象来调用数据库存储过程。`CallableStatement` 对象提供了设置其输入 (IN) 和输出 (OUT) 参数的方法，以及用来执行对存储过程的调用方法。方法返回



一个包含预编译的 SQL 语句的新的 `CallableStatement` 对象。调用方法如下：

```
CallableStatement prepareCall(String sql) throws SQLException
```

为了处理存储过程调用语句，此方法进行了优化。某些驱动程序可以在调用 `prepareCall` 方法后将调用语句发送给数据库；另一些则直至执行了 `CallableStatement` 对象后才可以发送。这对用户没有直接的影响。

使用 `prepareCall()` 方法返回的 `CallableStatement` 对象创建的结果记录集在默认情况下类型为 `TYPE_FORWARD_ONLY`，并带有 `CONCUR_READ_ONLY` 并发级别。



**提示** 参数 `sql` 中表示执行的调用存储过程的调用语句，其中可能包含一个或多个“?” 参数占位符。

`prepareCall()` 还有两种其他的使用形式，如下所示：

```
CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)
    throws SQLException;
CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency,
    int resultSetHoldability) throws SQLException;
```

这两种形式返回一个包含预编译的 SQL 语句的 `CallableStatement` 对象，该对象将生成具有给定类型和并发性的 `ResultSet` 对象。这两种形式与仅带 `sql` 参数的 `prepareCall()` 方法功能类似，但它允许重写默认结果记录集类型和并发级别。

`resultSetType` 参数可以是以下 `ResultSet` 常量之一：`ResultSet.TYPE_FORWARD_ONLY`、`ResultSet.TYPE_SCROLL_INSENSITIVE` 或 `ResultSet.TYPE_SCROLL_SENSITIVE`。

`resultSetConcurrency` 参数可以是以下 `ResultSet` 常量之一：`ResultSet.CONCUR_READ_ONLY` 或 `ResultSet.CONCUR_UPDATABLE`。

`resultSetHoldability` 参数可以是以下 `ResultSet` 常量之一：`ResultSet.HOLD_CURSORS_OVER_COMMIT` 或 `ResultSet.CLOSE_CURSORS_AT_COMMIT`。

#### (10) `prepareStatement()`

此方法创建一个 `PreparedStatement` 对象，以用这个对象来将参数化的 SQL 语句发送到数据库。带有输入 (IN) 参数或不带有输入 (IN) 参数的 SQL 语句都可以被预编译并存储在 `PreparedStatement` 对象中，然后可以使用此对象来多次执行该语句。调用方法如下：

```
PreparedStatement prepareStatement(String sql) throws SQLException
```

`prepareStatement` 可以对 SQL 语句作预编译处理，但前提是数据库驱动程序要支持预编译，`prepareStatement` 方法会将该语句发送给数据库进行预编译。有的数据库驱动程序可能不支持预编译，此时执行 `PreparedStatement` 对象之前无法将语句发送给数据库。

使用返回的 `PreparedStatement` 对象创建的结果记录集在默认情况下类型为 `TYPE_FORWARD_ONLY`，并带有 `CONCUR_READ_ONLY` 并发级别。

#### (11) `rollback()`

取消在当前事务中进行的所有更改，并释放当前 `Connection` 对象保存的所有数据库锁定。

`rollback()`方法只能在已禁用自动提交模式时使用。调用方法如下：

```
void rollback() throws SQLException
```

#### (12) `setAutoCommit()`

将当前连接的自动提交模式设置为给定状态。如果连接处于自动提交模式下，则将执行其所有 SQL 语句，并将这些语句作为单独的事务提交。否则，其 SQL 语句将成组地进入通过调用 `commit()` 方法或 `rollback()` 方法终止的事务中。默认情况下，新的连接处于自动提交模式下。调用方法如下：

```
void setAutoCommit(boolean autoCommit) throws SQLException
```

提交发生在语句完成或执行下一条语句时，以先发生的情况为准。在语句返回 `ResultSet` 对象的情况下，该语句在已检索完最后一行 `ResultSet` 对象或已关闭 `ResultSet` 对象时完成。在更复杂的情况下，单个语句可以返回多个结果和输出参数值。在这些情况下，提交发生在检索到所有结果和输出参数值后。

`autoCommit` 参数为 `true` 表示启用自动提交模式；为 `false` 表示禁用自动提交模式。

#### (13) `setHoldability()`

将使用当前 `Connection` 对象创建的 `ResultSet` 对象的可保存性(`holdability`)更改为给定的可保存性。参数 `holdability` 为 `ResultSet` 的可保存性常量，它是 `ResultSet.HOLD_CURSORS_OVER_COMMIT` 或 `ResultSet.CLOSE_CURSORS_AT_COMMIT` 之一。调用方法如下：

```
void setHoldability(int holdability) throws SQLException
```

#### (14) `setReadOnly()`

此方法用于将当前连接设置为只读模式。参数 `readOnly` 为 `true` 表示启用只读模式；为 `false` 表示禁用只读模式。调用方法如下：

```
void setReadOnly(boolean readOnly) throws SQLException
```

#### (15) `setTransactionIsolation()`

将当前 `Connection` 对象的事务隔离级别更改为给定的级别。参数 `level` 的值可以是以下 `Connection` 常量之一：`Connection.TRANSACTION_READ_UNCOMMITTED`、`Connection.TRANSACTION_READ_COMMITTED`、`Connection.TRANSACTION_REPEATABLE_READ` 或 `Connection.TRANSACTION_SERIALIZABLE`。调用方法如下：

```
void setTransactionIsolation(int level) throws SQLException
```

### 4. Statement

`Statement` 类对象代表 SQL 语句，可用于将 SQL 语句发送至数据库。其中存在三种 `Statement` 对象，一是 `Statement`，用来执行基本的 SQL 语句；二是 `PreparedStatement`，它从 `Statement` 继承而来，用于提供可以与查询信息一起预编译的语句；三是 `CallableStatement`，它继承自 `PreparedStatement`，用来执行数据库中的存储过程。后面会一一介绍。

`Statement` 用于执行静态 SQL 语句并返回它所生成结果的对象。在默认情况下，同一时刻每个 `Statement` 对象只能打开一个 `ResultSet` 对象，因此如果读取一个 `ResultSet` 对象与读取另一个交叉，则这两个对象必须是由不同的 `Statement` 对象生成的。如果存在某个语句打开的当前



ResultSet 对象，则 Statement 接口中的所有执行方法都会隐式关闭它。

Statement 接口常用的方法有如下一些。

(1) addBatch()

addBatch()方法用于将给定的 SQL 命令添加到此 Statement 对象的当前命令列表中，此后通过调用 executeBatch()方法可以批量执行此列表中的命令。参数 sql 通常为静态的 SQL INSERT 或 UPDATE 语句。调用方法如下：

```
void addBatch(String sql) throws SQLException
```

(2) cancel()

如果数据库管理系统和数据库驱动程序都支持终止 SQL 语句，则取消此 Statement 对象。一个线程可以使用此方法取消另一个线程正在执行的语句。调用方法如下：

```
void cancel() throws SQLException
```

(3) clearBatch()

清空此 Statement 对象的当前 SQL 命令列表。调用方法如下：

```
void clearBatch() throws SQLException
```

(4) close()

立即释放当前 Statement 对象的数据库和 JDBC 资源而不是等待该对象自动关闭时发生此操作。在使用完 Statement 对象后立即释放资源是一个好的习惯，这样可以避免对数据库资源的占用，但在已经关闭的 Statement 对象上调用 close()方法是无效的。调用方法如下：

```
void close() throws SQLException
```

(5) execute()

执行给定的 SQL 语句，该语句可能返回多个结果。在某些（不常见）情形下，单个 SQL 语句可能返回多个结果记录集合或更新影响的记录条数，execute()方法指示第一个结果的形式。如果第一个结果为 ResultSet 对象，则返回 true；如果其为更新计数或者不存在任何结果，则返回 false。执行 SQL 语句后，必须使用方法 getResultSet()或 getUpdateCount()来检索结果，使用 getMoreResults()来移动后续结果。调用方法如下：

```
boolean execute(String sql) throws SQLException
```

(6) executeBatch()

将一批命令提交给数据库来执行，如果全部命令执行成功，则返回更新计数组成的数组。返回数组的 int 元素的排序对应于批中的命令，批中的命令根据被添加到批中的顺序排序。调用方法如下：

```
int[] executeBatch() throws SQLException
```

方法 executeBatch()返回的数组中的元素可能为以下元素之一。

- 大于或等于零的数：指示成功处理了命令，是给出执行命令所影响数据库中行数的更新计数。
- SUCCESS\_NO\_INFO：指示成功执行了命令，但受影响的行数是未知的。如果批量更新中的命令之一无法正确执行，则此方法抛出 BatchUpdateException 异常，并且 JDBC 驱动程序可能继续处理批处理中的剩余命令，也可能不执行。无论如何，驱动程序的行为必

须与特定的数据库管理系统一致，要么始终继续处理命令，要么永远不继续处理命令。如果驱动程序在某一次失败后继续进行处理，则 `BatchUpdateException.getUpdateCounts` 方法返回的数组将包含的元素与批中存在的命令一样多，并且其中至少有一个元素将为 `EXECUTE_FAILED`。

- `EXECUTE_FAILED`: 指示未能成功执行命令，仅当命令失败后驱动程序继续处理命令时出现。如果发生数据库访问错误，或者驱动程序不支持批量语句将会抛出 `SQLException` 异常。

#### (7) `executeQuery()`

执行给定的 SQL 语句，该语句返回单个 `ResultSet` 对象。参数 `sql` 是要发送给数据库的 SQL 语句，通常为静态 SQL `SELECT` 语句。调用方法如下：

```
ResultSet executeQuery(String sql) throws SQLException
```

#### (8) `executeUpdate()`

执行给定 SQL 语句，该语句可能为 `INSERT`、`UPDATE` 或 `DELETE` 语句，或者不返回任何内容的 SQL 语句（如 SQL `DDL` 语句）。参数 `sql` 为 `INSERT`、`UPDATE` 或 `DELETE` 语句，或者不返回任何内容的 SQL 语句。`executeUpdate()` 方法返回 SQL 语句所影响的行数，如果是不返回任何内容的 SQL 语句或没有被影响的行数则返回 0。调用方法如下：

```
int executeUpdate(String sql) throws SQLException
```

#### (9) `getConnection()`

得到当前 `Statement` 对象的 `Connection` 对象。调用方法如下：

```
Connection getConnection() throws SQLException
```

#### (10) `getMaxFieldSize()`

获得可以为当前 `Statement` 对象所生成 `ResultSet` 对象中的字符和二进制列值返回的最大字节数。此限制仅应用于 `BINARY`、`VARBINARY`、`LONGVARBINARY`、`CHAR`、`VARCHAR` 和 `LONGVARCHAR` 列。如果超过了该限制，则会丢弃多出的数据。如果返回 0 则表示没有任何限制。调用方法如下：

```
int getMaxFieldSize() throws SQLException
```

#### (11) `getMaxRows()`

获得当前 `Statement` 对象生成的 `ResultSet` 对象可以包含的最大行数。如果超过了此限制，则会撤销多出的行。如果返回 0 则表示没有任何限制。调用方法如下：

```
int getMaxRows() throws SQLException
```

#### (12) `getMoreResults()`

移动到当前 `Statement` 对象的下一个结果，如果这个结果为 `ResultSet` 对象，则返回 `true`，并隐式关闭利用 `getResultSet()` 方法获取的所有当前 `ResultSet` 对象。调用方法如下：

```
boolean getMoreResults() throws SQLException
```

当以下表达式为 `true` 时没有更多结果：

```
// stmt 是一个 Statement 对象
((stmt.getMoreResults() != false) && (stmt.getUpdateCount() != -1))
```

如果下一个结果为 `ResultSet` 对象，则 `getMoreResults()` 方法返回 `true`；如果其为更新计数或不存在更多结果，则返回 `false`。



**(13) getResultSet()**

得到当前 **Statement** 对象执行 SQL 的结果 **ResultSet**，如果结果是更新计数或没有更多的结果则返回 **null**。调用方法如下：

```
ResultSet getResultSet() throws SQLException
```

**(14) getUpdateCount()**

得到当前 **Statement** 对象执行 SQL 的结果，结果以更新计数的形式返回；如果当前结果为 **ResultSet** 对象或没有更多结果，则返回 -1。调用方法如下：

```
int getUpdateCount() throws SQLException
```

**(15) isClosed()**

验证当前 **Statement** 对象是否已经被关闭，如果已经被关闭则返回 **true**，否则返回 **false**。调用方法如下：

```
boolean isClosed() throws SQLException
```

**(16) setMaxFieldSize()**

设置将字符或二进制值存储到给定 **ResultSet** 列中的最大字节数限制，此限制仅应用于 **BINARY**、**VARBINARY**、**LONGVARBINARY**、**CHAR**、**VARCHAR** 和 **LONGVARCHAR** 字段，如果超过了该限制，则会丢弃多出的数据。为了获得良好的可移植性，应该使用大于 256 的值。参数 **max** 是以字节为单位的新列大小限制，如果为 0 则表示没有任何限制。调用方法如下：

```
void setMaxFieldSize(int max) throws SQLException
```

**(17) setMaxRows()**

设置通过当前 **Statement** 对象所能获得的任何 **ResultSet** 对象都可以包含的最大行数限制，如果超过了该限制，则会撤销多出的行。参数 **max** 为新的最大行数限制，0 表示没有任何限制。调用方法如下：

```
void setMaxRows(int max) throws SQLException
```

## 5. PreparedStatement

**PreparedStatement** 用于表示预编译 SQL 语句的对象。SQL 语句被预编译并且存储在 **PreparedStatement** 对象中，然后可以使用此对象高效地多次执行该语句。

**PreparedStatement** 接口的声明情况如下：

```
public interface PreparedStatement extends Statement;
```

可见 **PreparedStatement** 接口扩展了 **Statement** 接口，下面仅列出不是继承自 **Statement** 接口的一个常用方法。

```
void setXxxx(int parameterIndex, xxxx x) throws SQLException;
```

将 SQL 语句中给定的第 **parameterIndex** 个参数设为 **x**。**Xxxx** 表数据类型，于是有如下常用的表示形式。

```
void setBoolean(int parameterIndex, boolean x) throws SQLException;
void setByte(int parameterIndex, byte x) throws SQLException;
void setBytes(int parameterIndex, byte x[]) throws SQLException;
void setDate(int parameterIndex, java.sql.Date x) throws SQLException;
```

```

void setDouble(int parameterIndex, double x) throws SQLException;
void setFloat(int parameterIndex, float x) throws SQLException;
void setInt(int parameterIndex, int x) throws SQLException;
void setLong(int parameterIndex, long x) throws SQLException;
void setObject(int parameterIndex, Object x) throws SQLException;
void setShort(int parameterIndex, short x) throws SQLException;
void setString(int parameterIndex, String x) throws SQLException;

```



**提示** `parameterIndex` 参数表示第几个参数，编号从 1 开始，也就是第一个参数的 `parameterIndex` 为 1，第二个参数的 `parameterIndex` 为 2。

## 6. CallableStatement

`CallableStatement` 是用于执行 SQL 存储过程的接口。JDBC API 提供了一个存储过程 SQL 转义语法，该语法允许对所有数据库管理系统使用标准方式调用存储过程，此转义语法有一个包含结果参数的形式和一个不包含结果参数的形式。参数是根据编号按顺序引用的，第一个参数的编号是 1。

```

{?= call <procedure-name>[<arg1>,<arg2>, ...]}
{call <procedure-name>[<arg1>,<arg2>, ...]}

```

存储过程的输入参数值是使用从 `PreparedStatement` 中继承的 `set` 方法设置的。在执行存储过程之前，必须注册所有输出参数的类型；它们的值是在执行后通过此类提供的 `get` 方法获得的。注册输出参数使用 `registerOutParameter()` 方法，有如下的几种形式。

```

void registerOutParameter(int parameterIndex, int sqlType) throws SQLException;
void registerOutParameter(int parameterIndex, int sqlType, int scale)
throws SQLException

```

以上方法的功能是按顺序位置 `parameterIndex` 将输出参数注册为 JDBC 类型 `sqlType`。由 `sqlType` 指定的输出参数的 JDBC 类型确定必须用 `getXxxx()` 方法来读取该参数值的 Java 类型。如果预期返回给此输出参数的 JDBC 类型是取决于此特定数据库的，则 `sqlType` 应该是 `java.sql.Types.OTHER`。方法 `getObject(int)` 检索该值。

`java.sql.Types` 类中定义了用于标识一般 SQL 类型（称为 JDBC 类型）的常量的类。常用的有：`ARRAY`、`BIGINT`、`BINARY`、`BIT`、`BLOB`、`BOOLEAN`、`CHAR`、`CLOB`、`DATA`、`DECIMAL`、`DOUBLE`、`FLOAT`、`INTEGER`、`NUMERIC`、`OTHER`、`TIME`、`TIMESTAMP`、`VARCHAR` 等。

要得到存储过程的返回结果记录集一般使用 `PreparedStatement` 类的 `executeQuery()` 方法，而不必去注册输出参数，`executeQuery()` 方法也是最为常用的得到输出参数值的方法。

`CallableStatement` 可以返回一个 `ResultSet` 对象或多个 `ResultSet` 对象。多个 `ResultSet` 对象是使用从 `Statement` 中继承的操作处理的。

为了获得良好的可移植性，某一调用的 `ResultSet` 对象和更新计数应该在获得输出参数的值之前处理。`CallableStatement` 接口的声明情况如下。

```

public interface CallableStatement extends PreparedStatement;

```

可见 `CallableStatement` 接口扩展了 `PreparedStatement` 接口，而 `PreparedStatement` 接口扩展了 `Statement` 接口。



## 7. ResultSet

**ResultSet** 表示数据库结果记录集的数据表,通常通过执行查询数据库的语句生成。**ResultSet** 对象具有指向其当前数据行的指针。最初指针被置于第一行之前, **next()** 方法将指针移动到下一行; 因为该方法在 **ResultSet** 对象中没有下一行时返回 **false**, 所以可以在 **while** 循环中使用它来迭代结果记录集。

默认情况下, **ResultSet** 对象不可更新, 且仅有一个向前移动的指针, 因此只能迭代它一次, 并且只能按从第一行到最后一行的顺序进行。可以生成可滚动或可更新的 **ResultSet** 对象。以下代码片段 (其中 **con** 为有效的数据库连接 **Connection** 对象) 演示了如何生成可滚动且不受其他更新影响的、可更新的结果记录集。

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT 字段名 1, 字段名 2 FROM 表名");
//rs 是可滚动且不受其他更新影响的、可更新的结果记录集
```

**ResultSet** 接口提供用于从当前行检索列值的获取方法 (**getBoolean()**、**getLong()**等), 可以使用列的索引编号或列的名称检索值, 一般地使用列索引较为高效, 列从 1 开始编号。为了获得良好的可移植性, 应该按从左到右的顺序读取每行中的结果记录集列, 而且每列只能读取一次。

对于获取方法, **JDBC** 驱动程序尝试将基础数据转换为在获取方法中指定的 **Java** 类型, 并返回适当的 **Java** 值。**JDBC** 规范中提供了允许从 **SQL** 类型到供 **ResultSet** 获取方法使用的 **Java** 类型的映射关系, 如表 6-2 所示。

表 6-2 JDBC 规范 (3.0) 数据类型转换表

JDBC 类型	Java 类型	JDBC 类型	Java 类型
CHAR	java.lang.String	BINARY	byte[]
VARCHAR	java.lang.String	VARBINARY	byte[]
LONGVARCHAR	java.lang.String	LONGVARBINARY	byte[]
NUMERIC	java.math.BigDecimal	DATE	java.sql.Date
DECIMAL	java.math.BigDecimal	TIME	java.sql.Time
BIT	boolean	TIMESTAMP	java.sql.Timestamp
BOOLEAN	boolean	CLOB	java.sql.Clob
TINYINT	byte	BLOB	java.sql.Blob
SMALLINT	short	ARRAY	java.sql.Array
INTEGER	int	DISTINCT	(基础类型的映射)
BIGINT	long	STRUCT	java.sql.Struct
REAL	float	REF	java.sql.Ref
FLOAT	double	DATALINK	java.net.URL
DOUBLE	double	JAVA_OBJECT	(基础 Java 类)

用作获取方法的输入的列名称不区分大小写。用列名称调用获取方法时, 如果多个列具有这一名称, 则返回第一个匹配列的值。列名称选项在生成结果记录集的 **SQL** 查询中使用列名称时使用, 对于没有在查询中显示命名的列, 最好使用列编号, 因为如何使用列名称无法保证名

称实际所指的就是预期的列。

可以用以下两种方式进行更新。

(1) 更新当前行中的列值。在可滚动的 `ResultSet` 对象中,可以向前和向后移动指针,将其置于绝对位置或相对于当前行的位置,以下代码片段更新 `ResultSet` 对象 `rs` 的第五行中的 `NAME` 列,然后使用 `updateRow()` 更新用于生成结果记录集 `rs` 的数据源表。

```
rs.absolute(5); //将指针移动到结果记录集 rs 的第 5 条记录
//将第 5 条记录的 NAME 字段值设为"dengjiarong"
rs.updateString("NAME", "dengjiarong");
rs.updateRow(); // 更新数据库中的数据
```

(2) 将列值插入到插入行中。可更新的 `ResultSet` 对象具有一个与其关联的特殊行,该行用作构建要插入的行的暂存区域。以下代码片段将指针移动到插入行,构建一个 3 列的行,并使用 `insertRow()` 方法将其插入到结果记录集 `rs` 和数据源表中。

```
rs.moveToInsertRow(); //移动当前记录指针标到 insert row
rs.updateString(1, "dengjiarong"); //将 insert row 中第 1 列的值更新为 "dengjiarong"
rs.updateInt(2,35); //将 insert row 中第 2 列的值更新为 "35"
rs.updateBoolean(3, true); //将 insert row 中第 3 列的值更新为 "true"
rs.insertRow(); //更新数据
rs.moveToCurrentRow();
```



**提示** `insert row` 是结果记录集中的一个特殊的记录,它实质上是构造的一个数据缓冲区。

当生成 `ResultSet` 对象的 `Statement` 对象关闭、重新执行或用来从多个结果的序列检索下一个结果时, `ResultSet` 对象会自动关闭。

`ResultSet` 有 10 个常量属性,其对应的值如表 6-3 所示。

表 6-3 `ResultSet` 的常量属性的值

常 量	值
<code>HOLD_CURSORS_OVER_COMMIT</code>	1
<code>CLOSE_CURSORS_AT_COMMIT</code>	2
<code>FETCH_FORWARD</code>	1000
<code>FETCH_REVERSE</code>	1001
<code>FETCH_UNKNOWN</code>	1002
<code>TYPE_FORWARD_ONLY</code>	1003
<code>TYPE_SCROLL_INSENSITIVE</code>	1004
<code>TYPE_SCROLL_SENSITIVE</code>	1005
<code>CONCUR_READ_ONLY</code>	1007
<code>CONCUR_UPDATABLE</code>	1008

`HOLD_CURSORS_OVER_COMMIT` 表示调用 `Connection.commit()` 方法时不应关闭 `ResultSet` 对象。`CLOSE_CURSORS_AT_COMMIT` 表示调用 `Connection.commit()` 方法时应该关闭 `ResultSet` 对象。

`FETCH_FORWARD` 表示将按正向(即从第一个到最后一个)处理结果记录集中的行。



**FETCH\_REVERSE** 表示将按反向（即从最后一个到第一个）处理结果记录集中的行。  
**FETCH\_UNKNOWN** 表示结果记录集中行的处理顺序未知。

**TYPE\_FORWARD\_ONLY** 表示指针只能向前移动的 **ResultSet** 对象的类型。  
**TYPE\_SCROLL\_INSENSITIVE** 表示可滚动但通常不受其他更改影响的 **ResultSet** 对象的类型。  
**TYPE\_SCROLL\_SENSITIVE** 表示可滚动并且通常受其他更改影响的 **ResultSet** 对象的类型。

**CONCUR\_READ\_ONLY** 表示不可以更新的 **ResultSet** 对象的并发模式。

**CONCUR\_UPDATABLE** 表示可以更新的 **ResultSet** 对象的并发模式。

**ResultSet** 接口的常用方法有如下一些。

#### (1) absolute()

**absolute()**方法用于将指针移动到此 **ResultSet** 对象的给定行编号。如果行编号为正，则将指针移动到相对于结果记录集开头的给定行编号，第一行为行 1，第二行为行 2，依此类推。调用效果如下：

```
boolean absolute( int row ) throws SQLException
```

如果给定行编号为负，则将指针移动到相对于结果记录集末尾的绝对行位置。例如，调用方法 **absolute(-1)**将指针置于最后一行；调用方法 **absolute(-2)**将指针移动到倒数第二行，依此类推。

如果试图将指针置于结果记录集的第一行或最后一行之外，相应地将导致指针位于第一行之前或最后一行之后。



**提示** 调用 **absolute(1)**等效于调用 **first()**。调用 **absolute(-1)**等效于调用 **last()**。

参数 **row** 是想要移动到的行的编号。正的编号指示从结果记录集开头开始计数的行编号；负的编号指示从结果记录集末尾开始计数的行编号。如果调用 **absolute()**方法后指针位于结果记录集上，则返回 **true**；否则返回 **false**。



**提示** 如果结果记录集类型为 **TYPE\_FORWARD\_ONLY** 调用 **absolute()**方法将抛出 **SQLException** 异常。

#### (2) afterLast()

将指针移动到此 **ResultSet** 对象的末尾，正好位于最后一行之后。如果结果记录集中不包含任何行，则此方法无效。调用方法如下：

```
void afterLast() throws SQLException
```

#### (3) beforeFirst()

将指针移动到此 **ResultSet** 对象的开头，正好位于第一行之前。如果结果记录集中不包含任何行，则此方法无效。调用方法如下：

```
void beforeFirst() throws SQLException
```

#### (4) cancelRowUpdates()

取消对 **ResultSet** 对象中的当前行所作的更新。此方法在调用更新方法之后，但在调用

`updateRow()`方法之前调用才可以回滚对行所作的更新。如果没有进行任何更新或者已经调用 `updateRow()`方法, 则此方法无效。调用方法如下:

```
void cancelRowUpdates() throws SQLException
```

#### (5) `close()`

立即释放此 `ResultSet` 对象的数据库和 JDBC 资源。当生成 `ResultSet` 对象的 `Statement` 对象关闭、重新执行或用来从多个结果的序列检索下一个结果时, 该 `Statement` 对象会自动关闭 `ResultSet` 对象。垃圾回收 `ResultSet` 对象时它也会自动关闭。调用方法如下:

```
void close() throws SQLException
```

#### (6) `deleteRow()`

从当前 `ResultSet` 对象和底层数据库中删除当前行。指针不位于插入行上时不能调用此方法。调用方法如下:

```
void deleteRow() throws SQLException
```

#### (7) `first()`

将指针移动到当前 `ResultSet` 对象的第一行。如果指针位于有效行, 则返回 `true`; 如果结果记录集中不存在任何行, 则返回 `false`。调用方法如下:

```
boolean first() throws SQLException
```



**提示** 如果结果记录集类型为 `TYPE_FORWARD_ONLY` 调用 `first()`方法将抛出 `SQLException` 异常。

#### (8) `getXxxx(int columnIndex)`

此方法以 Java 中的数据类型来得到 `ResultSet` 对象中当前行的第 `columnIndex` 列的值, 第一个列的 `columnIndex` 为 1, 第二个列的 `columnIndex` 为 2, 依此类推。调用方法如下:

```
xxxx getXxxx(int columnIndex) throws SQLException
```

如果列值为 SQL `null` 则返回 `null`。此方法根据 Java 数据类型的不同, 主要有如下的表现形式:

```
boolean getBoolean(int columnIndex) throws SQLException;
byte getByte(int columnIndex) throws SQLException;
byte[] getBytes(int columnIndex) throws SQLException;
Date getDate(int columnIndex) throws SQLException;
double getDouble(int columnIndex) throws SQLException;
float getFloat(int columnIndex) throws SQLException;
int getInt(int columnIndex) throws SQLException;
long getLong(int columnIndex) throws SQLException;
object getObject(int columnIndex) throws SQLException;
short getShort(int columnIndex) throws SQLException;
String getString(int columnIndex) throws SQLException;
```



**提示** `getDate()`方法返回的数据类型为 `java.sql.Date`, 而非 `java.util.Date`。



**(9) getXxxx(String columnName)**

此方法以 Java 中的数据类型来得到 **ResultSet** 对象中当前行的列名为 **columnName** 列的值。调用方法如下：

```
xxxx getXxxx(String columnName) throws SQLException
```

如果列值为 SQL null 则返回 null。此方法根据 Java 数据类型的不同，主要有如下的表现形式。

```
boolean getBoolean(String columnName) throws SQLException;
byte getByte(String columnName) throws SQLException;
byte[] getBytes(String columnName) throws SQLException;
Date getDate(String columnName) throws SQLException;
double getDouble(String columnName) throws SQLException;
float getFloat(String columnName) throws SQLException;
int getInt(String columnName) throws SQLException;
long getLong(String columnName) throws SQLException;
object getObject(String columnName) throws SQLException;
short getShort(String columnName) throws SQLException;
String getString(String columnName) throws SQLException;
```

**(10) getConcurrency()**

得到当前 **ResultSet** 对象的并发模式，值为 **ResultSet.CONCUR\_READ\_ONLY** 或 **ResultSet.CONCUR\_UPDATABLE**。调用方法如下：

```
int getConcurrency() throws SQLException
```

**(11) getRow()**

得到当前行的编号。第一行为 1 号，第二行为 2 号，依此类推。如果不存在当前行则返回 0。调用方法如下：

```
int getRow() throws SQLException
```

**(12) getType()**

得到当前 **ResultSet** 对象的类型，返回值 **ResultSet.TYPE\_FORWARD\_ONLY**、**ResultSet.TYPE\_SCROLL\_INSENSITIVE** 或 **ResultSet.TYPE\_SCROLL\_SENSITIVE** 中的一个。调用方法如下：

```
int getType() throws SQLException
```

**(13) insertRow()**

将插入行的内容插入到当前 **ResultSet** 对象和数据库中，此时当前指针必须位于插入行上。如果发生数据库访问错误，或者当前指针不位于插入行上时调用此方法，或者插入行中所有不可为 null 的列中还存在未分配值的列，则会抛出 **SQLException** 异常。调用方法如下：

```
void insertRow() throws SQLException
```

**(14) isAfterLast()**

判断当前指针是否位于 **ResultSet** 对象的最后一行之后，是则返回 true，否则返回 false。调用方法如下：

```
boolean isAfterLast() throws SQLException
```

**(15) isBeforeFirst()**

判断当前指针是否位于 **ResultSet** 对象的第一行之前，是则返回 true，否则返回 false。调

用方法如下：

```
boolean isBeforeFirst() throws SQLException
```

#### (16) isFirst()

判断当前指针是否位于 **ResultSet** 对象的第一行，是则返回 **true**，否则返回 **false**。调用方法如下：

```
boolean isFirst() throws SQLException
```

#### (17) isLast()

判断当前指针是否位于 **ResultSet** 对象的最后一行，是则返回 **true**，否则返回 **false**。调用方法如下：

```
boolean isLast() throws SQLException
```

#### (18) last()

将指针移动到当前 **ResultSet** 对象的最后一行。如果指针位于有效行，则返回 **true**；如果结果记录集中不存在任何行，则返回 **false**。调用方法如下：

```
boolean last() throws SQLException
```



**提示** 如果结果记录集类型为 **TYPE\_FORWARD\_ONLY** 调用 **first()** 方法将抛出 **SQLException** 异常。

#### (19) moveToInsertRow()

将指针移动到插入行。将指针置于插入行上时，当前的指针位置会被记住。插入行是一个与可更新结果集相关联的特殊行。它实际上是一个缓冲区，在将行插入到结果集前可以通过调用更新方法在其中构造新行。当指针位于插入行上时，仅能调用更新方法、获取方法以及 **insertRow()** 方法。每次在调用 **insertRow()** 之前调用此方法时，必须为结果集中的所有列分配值。在对列值调用获取方法之前，必须调用更新方法。如果发生数据库访问错误或者结果集不可更新将会抛出 **SQLException** 异常。调用方法如下：

```
void moveToInsertRow() throws SQLException
```

#### (20) next()

将指针从当前位置下移一行。**ResultSet** 指针最初位于第一行之前；第一次调用 **next()** 方法使第一行成为当前行；第二次调用使第二行成为当前行，依此类推。如果新的当前行有效，则返回 **true**；如果不存在下一行，则返回 **false**。调用方法如下：

```
boolean next() throws SQLException
```

#### (21) previous()

将当前指针移动到 **ResultSet** 对象的上一行。如果指针位于有效行上，则返回 **true**；如果它不在结果集中，则返回 **false**。如果发生数据库访问错误或者结果集类型为 **TYPE\_FORWARD\_ONLY** 将会抛出 **SQLException** 异常。调用方法如下：

```
boolean previous() throws SQLException
```



### (22) relative()

按相对行数（或正或负）移动指针。试图移动到结果集的第一行或最后一行之外，会将指针置于第一行之前或最后一行之后。调用 `relative(0)` 有效，但是不更改指针位置。参数 `row` 指定从当前行开始移动的行数，正数表示指针向前移动，负数表示指针向后移动。如果移动后的指针位于行上，则返回 `true`，否则返回 `false`。调用方法如下：

```
boolean relative(int rows) throws SQLException
```



**提示** 调用方法 `relative(1)` 等效于调用方法 `next()`，而调用方法 `relative(-1)` 等效于调用方法 `previous()`。

### (23) updateXxxx(int columnIndex, xxxx x)

用指定的 Java 数据类型更新当前行的第 `columnIndex` 列的值。调用方法如下：

```
void updateXxxx(int columnIndex, xxxx x) throws SQLException
```

调用此方法并不会更新底层数据库；更新数据库要调用 `updateRow()` 或 `insertRow()` 方法。此方法根据 Java 数据类型的不同，主要有如下的表现形式。

```
void updateBoolean(int columnIndex, boolean b) throws SQLException;
void updateByte(int columnIndex, byte b) throws SQLException;
void updateBytes(int columnIndex, byte[] b) throws SQLException;
void updateDate(int columnIndex, Date b) throws SQLException;
void updateDouble(int columnIndex, double b) throws SQLException;
void updateFloat(int columnIndex, float b) throws SQLException;
void updateInt(int columnIndex, int b) throws SQLException;
void updateObject (int columnIndex, object b) throws SQLException;
void updateShort(int columnIndex, short b) throws SQLException;
void updateString(int columnIndex, String b) throws SQLException;
```

### (24) updateXxxx(String columnName, xxxx x)

用指定的 Java 数据类型更新当前行的列 `columnName` 的值。调用方法如下：

```
void updateXxxx(String columnName, xxxx x) throws SQLException
```

调用此方法并不会更新底层数据库；更新数据库要调用 `updateRow()` 或 `insertRow()` 方法。此方法根据 Java 数据类型的不同，主要有如下的表现形式。

```
void updateBoolean(String columnName, boolean b) throws SQLException;
void updateByte(String columnName, byte b) throws SQLException;
void updateBytes(String columnName, byte[] b) throws SQLException;
void updateDate(String columnName, Date b) throws SQLException;
void updateDouble(String columnName, double b) throws SQLException;
void updateFloat(String columnName, float b) throws SQLException;
void updateInt(String columnName, int b) throws SQLException;
void updateObject (String columnName, object b) throws SQLException;
void updateShort(String columnName, short b) throws SQLException;
void updateString(String columnName, String b) throws SQLException;
```

### (25) updateRow()

将 `ResultSet` 对象的当前行的新内容更新底层数据库。指针不位于插入行上时不能调用此方法。如果发生数据库访问错误或者指针不位于插入行上时调用了此方法则会抛出 `SQLException`

异常。调用方法如下：

```
void updateRow() throws SQLException
```

## 6.3 查询记录

前面已经介绍了有关数据库操作的 Java 类及其方法,下面就要用这些方法来实现数据库的操作。

### 6.3.1 顺序查询

#### 【实例 6-1】顺序查询数据库表中的数据

##### 1. JDBC-ODBC 方式

假设已按照 6.2.3 节中的步骤建立了连接到本地 SQL Server 数据库的 ODBC 系统数据源 testDatabase。

##### selectUserTable1.jsp

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.sql.*" %>
<HTML>
<BODY>
  <% Connection con;
    Statement sql;
    ResultSet rs;
    try {
      con=DriverManager.getConnection("jdbc:odbc:testDataBase","sa","123");
      sql=con.createStatement();
      rs=sql.executeQuery("SELECT * FROM userTable");
      out.print("<Table Border>");
      out.print("<TR><td colspan=8 align=center>用户数据</td></tr>");
      out.print("<TR>");
      out.print("<Td width=100 >用户 ID 号</td>");
      out.print("<Td width=50 >用户名</td>");
      out.print("<Td width=100>用户真实姓名</td>");
      out.print("<Td width=50>年龄</td>");
      out.print("<Td width=50>性别</td>");
      out.print("<Td width=100>联系地址</td>");
      out.print("<Td width=100>联系电话</td>");
      out.print("<Td width=100>添加时间</td>");
      out.print("</TR>");
      while(rs.next()){
        out.print("<TR>");
        out.print("<TD >" + rs.getLong(1) + "</TD>");
        out.print("<TD >" + rs.getString(2) + "</TD>");
        out.print("<TD >" + rs.getString(4) + "</TD>");
        out.print("<TD >" + rs.getInt("user_age") + "</TD>");
```



```

        out.print("<TD >" + rs.getString("user_sex") + "</TD>");
        out.print("<TD >" + rs.getString("user_address") + "</TD>");
        out.print("<TD >" + rs.getString("user_telephone") + "</TD>");
        out.print("<TD >" + rs.getString("add_time") + "</TD>");
        out.print("</TR>");
    }
    out.print("</Table>");
    con.close();
} catch (SQLException e1) {
    out.print(e1);
}
%>
</BODY>
</HTML>

```

该程序先用类方法 `DriverManager.getConnection("jdbc:odbc:testDB")` 得到连接对象，用连接对象的 `createStatement()` 方法创建 SQL 语句对象，通过语句对象来执行 SQL 查询，得到结果记录集后，用 `next()` 方法移动记录指针，从而输出所有记录。



**注意** 在文件头部要导入 `java.sql.*` 包。

该程序的运行结果如图 6-10 所示。

用户ID号	用户名	用户真实姓名	年龄	性别	联系地址	联系电话	添加时间
1	jiaorong	邓佳容	30	女	湖南长沙高新区	8993421	2009-04-25 13:43:35.107
2	xingyu	刘新宇	34	男	湖南现代物流学院	3423343	2009-04-25 13:49:57.513
3	chengyun	程云	44	男	长沙中南大学	4433232	2009-04-25 13:50:30.420
4	xiaoming	刘小敏	34	女	首都师范大学	010-22332232	2009-04-25 14:21:34.077
6	liuyiping	刘一平	44	男	上海交通大学	029-3342233	2009-04-25 15:03:56.950
7	dengdt	邓地天	30	男	上海交通大学工学院	029-33422333	2009-04-25 15:04:32.560

图 6-10 顺序查询数据库表中的数据

## 2. JDBC 方式

假定已按 6.2.4 的方法安装了 JDBC 驱动。程序代码如下：

### selectUserTable2.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<HTML>
<BODY>
<%
    Connection con;
    Statement sql;
    ResultSet rs;
    try {
        con=DriverManager.getConnection("jdbc:sqlserver:"+
            "///localhost:1433;DatabaseName=testDatabase", "sa", "123");
        sql=con.createStatement();
    }

```

```

rs=sql.executeQuery("SELECT * FROM userTable");
out.print("<Table Border>");
out.print("<TR><td colspan=8 align=center>用户数据</td></tr>");
out.print("<TR>");
    out.print("<Td width=100 >用户 ID 号</td>");
    out.print("<Td width=50 >用户名</td>");
    out.print("<Td width=100>用户真实姓名</td>");
    out.print("<Td width=50>年龄</td>");
    out.print("<Td width=50>性别</td>");
    out.print("<Td width=100>联系地址</td>");
    out.print("<Td width=100>联系电话</td>");
    out.print("<Td width=100>添加时间</td>");
out.print("</TR>");
while(rs.next()){
    out.print("<TR>");
    out.print("<TD >"+rs.getLong(1)+"</TD>");
    out.print("<TD >"+rs.getString(2)+"</TD>");
    out.print("<TD >"+rs.getString(4)+"</TD>");
    out.print("<TD >"+rs.getInt("user_age")+"</TD>");
    out.print("<TD >"+rs.getString("user_sex")+"</TD>");
    out.print("<TD >"+rs.getString("user_address")+"</TD>");
    out.print("<TD >"+rs.getString("user_telephone")+"</TD>");
    out.print("<TD >"+rs.getString("add_time")+"</TD>");
    out.print("</TR>");
}
out.print("</Table>");
con.close();
}catch(SQLException e1) {
    out.print(e1);
}
%>
</BODY>
</HTML>

```

读者可能已经发现,这个程序中的代码与 JDBC-ODBC 方式的程序代码相比, Class.forName() 方法已经不需要了,且只是 DriverManager.getConnection() 方法的参数上有所区别,可见使用起来也简便多了。在 DriverManager.getConnection() 方法中,因为 JDBC-ODBC 方式在建立数据源时已经输入过用户名和密码,故不必在方法参数中给出,而 JDBC 中则要给出。

调试程序后可以发现两种连接方式的程序运行结果是一样的。基于此,在以后的程序代码中仅会演示 JDBC 方式。有兴趣的读者可自行调试 JDBC-ODBC 方式。

### 6.3.2 移动查询

移动查询就是要在记录集中上下移动记录指针,或定位到指定的行,以输出程序员想要输出的记录数据。

#### 【实例 6-2】移动查询

selectUserTable3.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
```



```

<%@ page import="java.sql.*" %>
<HTML>
<BODY>
<% Connection con;
Statement sql;
ResultSet rs;
try {
con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
":1433;DatabaseName=testDatabase","sa","123");
sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
rs=sql.executeQuery("SELECT * FROM userTable");
rs.last();//移动到最后一行
int rowNum=rs.getRow();//得到最后一行的行号
out.print("<Table Border>");
out.print("<TR<td colspan=8 align=center>用户数据(逆序输出
共"+rowNum+"条记录</td></tr>");
out.print("<TR>");
out.print("<Td width=100 >"+ "用户 ID 号</td>");
out.print("<Td width=50 >"+ "用户名</td>");
out.print("<Td width=100>"+ "用户真实姓名</td>");
out.print("<Td width=50>"+ "年龄</td>");
out.print("<Td width=50>"+ "性别</td>");
out.print("<Td width=100>"+ "联系地址</td>");
out.print("<Td width=100>"+ "联系电话</td>");
out.print("<Td width=100>"+ "添加时间</td>");
out.print("</TR>");
rs.afterLast();//移动到最后一行的后面
while(rs.previous()){
out.print("<TR>");
out.print("<TD >"+rs.getLong(1)+"</TD>");
out.print("<TD >"+rs.getString(2)+"</TD>");
out.print("<TD >"+rs.getString(4)+"</TD>");
out.print("<TD >"+rs.getInt("user_age")+"</TD>");
out.print("<TD >"+rs.getString("user_sex")+"</TD>");
out.print("<TD >"+rs.getString("user_address")+"</TD>");
out.print("<TD >"+rs.getString("user_telephone")+"</TD>");
out.print("<TD >"+rs.getString("add_time")+"</TD>");
out.print("</TR>");
}
rs.absolute(2);//移动到第 2 行
out.print("<TR<td colspan=8 align=center>指定输出第 2 个用户的数据</td></tr>");
out.print("<TR>");
out.print("<TD >"+rs.getLong(1)+"</TD>");
out.print("<TD >"+rs.getString(2)+"</TD>");
out.print("<TD >"+rs.getString(4)+"</TD>");
out.print("<TD >"+rs.getInt("user_age")+"</TD>");
out.print("<TD >"+rs.getString("user_sex")+"</TD>");
out.print("<TD >"+rs.getString("user_address")+"</TD>");
out.print("<TD >"+rs.getString("user_telephone")+"</TD>");
out.print("<TD >"+rs.getString("add_time")+"</TD>");
out.print("</TR>");
}
}

```

```

        out.print("</Table>");
        con.close();
    }
    catch(SQLException e1)
    {
        out.print(e1);
    }
    %>
</BODY>
</HTML>

```

该程序实现了记录的逆序输出和指定显示出第2行记录的数据。数据库连接对象 con 的方法 createStatement (ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_READ\_ONLY) 声明,生成的语句对象执行查询的结果集是可以上下移动的,故可以灵活地移动记录指针。程序的运行结果如图 6-11 所示。



用户ID号	用户名	用户真实姓名	年龄	性别	联系地址	联系电话	添加时间
3	chengyun	程云	44	男	长沙中南大学	4433232	2009-04-25 13:50:30.42
2	xingyu	刘新宇	34	男	湖南现代物流学院	3423343	2009-04-25 13:49:57.513
1	jiarong	邓佳容	11	女	湖南长沙高新区	8993421	2009-04-25 13:43:35.107

2	xingyu	刘新宇	34	男	湖南现代物流学院	3423343	2009-04-25 13:49:57.513
---	--------	-----	----	---	----------	---------	-------------------------

图 6-11 随机查询

### 6.3.3 参数查询

这种查询是指由客户端提交查询的条件,即为查询的参数,再根据这个参数构造查询 SQL 语句,并显示查询的结果。

#### 【实例 6-3】参数查询

##### selectUserTable4.jsp

```

<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.sql.*" %>
<%!
public String codeToString(String str){//处理中文字符串的函数
    String s=str;
    try{
        byte tempB[]=s.getBytes("ISO-8859-1");
        s=new String(tempB);
        return s;
    }catch(Exception e){
        return s;
    }
}
%>

```



```

<%//构造查询 SQL 语句
String sqlString=null;//SQL 语句
String sex=codeToString(request.getParameter("sex"));
if(sex==null||sex.trim().length()==0)
    sqlString=new String("SELECT * FROM userTable");
else
    sqlString=new String("SELECT * FROM userTable
    where user_sex='"+sex.trim()+"'");
%>
<HTML>
<BODY>
<% Connection con;
Statement sql;
ResultSet rs;
try {
    con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
        ":1433;DatabaseName=testDatabase","sa","123");
    sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
    rs=sql.executeQuery(sqlString);
    rs.last();
    int rowNumber=rs.getRow();
    out.print("<Table Border>");
    out.print("<form action=selectUserTable4.jsp method=post>");
    out.print("<TR><td>查询性别</td>");
    out.print("<td colspan=7>");
    out.print("<select name=sex>");
    out.print("<option value='' selected>所有</option>");
    out.print("<option value=男>男</option>");
    out.print("<option value=女>女</option></select>");
    out.print("<input type=submit value=提交>");
    out.print("</td></tr></form>");
    out.print("<TR><td colspan=8 align=center>用户数据
    (共"+rowNumber+"条记录)</td></tr>");
    out.print("<TR>");
    out.print("<Td width=100 >"+ "用户 ID 号</td>");
    out.print("<Td width=50 >"+ "用户名</td>");
    out.print("<Td width=100>"+ "用户真实姓名</td>");
    out.print("<Td width=50>"+ "年龄</td>");
    out.print("<Td width=50>"+ "性别</td>");
    out.print("<Td width=100>"+ "联系地址</td>");
    out.print("<Td width=100>"+ "联系电话</td>");
    out.print("<Td width=100>"+ "添加时间</td>");
    out.print("</TR>");
    rs.beforeFirst();
    while(rs.next())
    { out.print("<TR>");
        out.print("<TD >"+rs.getLong(1)+"</TD>");
        out.print("<TD >"+rs.getString(2)+"</TD>");
        out.print("<TD >"+rs.getString(4)+"</TD>");
        out.print("<TD >"+rs.getInt("user_age")+"</TD>");
        out.print("<TD >"+rs.getString("user_sex")+"</TD>");
        out.print("<TD >"+rs.getString("user_address")+"</TD>");
    }
} catch (Exception e) {
    out.print("<TR><td colspan=8>发生异常:<br>"+e.getMessage()+"</td></tr>");
}
%>

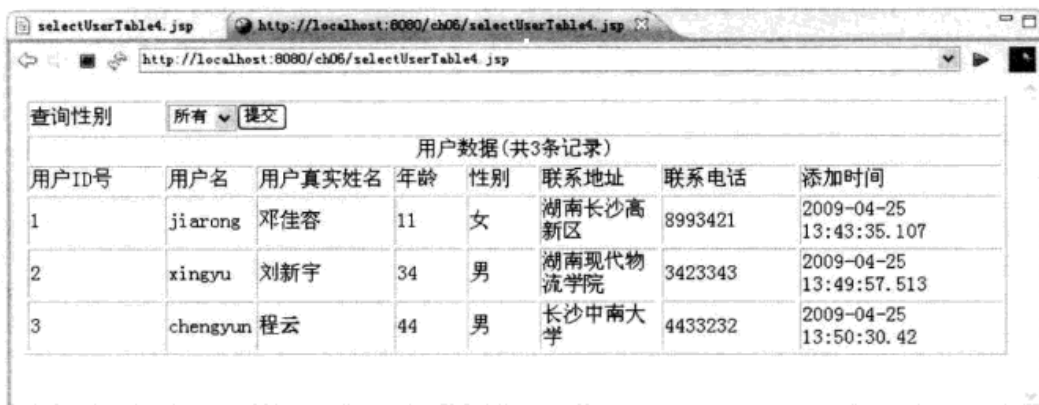
```

```

        out.print("<TD>"+rs.getString("user_telephone")+"</TD>");
        out.print("<TD>"+rs.getString("add_time")+"</TD>");
        out.print("</TR>");
    }
    out.print("</Table>");
    con.close();
} catch(SQLException e1) {
    out.print(e1);
}
%>
</BODY>
</HTML>

```

该程序实现了按性别查询用户信息。在程序的开始处加入了中文字符处理函数 `codeToString`，对于用户提交的数据要进行编码转换；接收数据后，看数据是否为空或是否空串，如果是则不必设置 SQL 语句的查询条件，如果不是则要设置查询条件。程序的运行结果如图 6-12 所示。



用户数据(共3条记录)							
用户ID号	用户名	用户真实姓名	年龄	性别	联系地址	联系电话	添加时间
1	jiarong	邓佳容	11	女	湖南长沙高新区	8993421	2009-04-25 13:43:35.107
2	xingyu	刘新宇	34	男	湖南现代物流学院	3423343	2009-04-25 13:49:57.513
3	chengyun	程云	44	男	长沙中南大学	4433232	2009-04-25 13:50:30.42

图 6-12 参数查询

### 6.3.4 模糊查询

#### 【实例 6-4】模糊查询

##### selectUserTable5.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<%!
public String codeToString(String str){//处理中文字符串的函数
    String s=str;
    try{
        byte tempB[]=s.getBytes("ISO-8859-1");
        s=new String(tempB);
        return s;
    } catch(Exception e){
        return s;
    }
}
%>

```



```

%>
<%//构造查询 SQL 语句
String sqlString=null;//SQL 语句
String username= codeToString(request.getParameter("username"));
if(username==null||username.trim().length()==0)
    sqlString=new String("SELECT * FROM userTable");
else
    sqlString=new String("SELECT * FROM userTable where user_true_name
        like '%" +username.trim()+"%'");
%>
<HTML>
<BODY>
<% Connection con;
Statement sql;
ResultSet rs;
try {
    con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
        ":1433;DatabaseName=testDatabase","sa","123");
    sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
    rs=sql.executeQuery(sqlString);
    rs.last();
    int rowNumber=rs.getRow();
    out.print("<Table Border>");
    out.print("<form action=selectUserTable5.jsp method=post>");
    out.print("<TR><td>查询姓名</td>");
    out.print("<td colspan=7>");
    out.print("<input type=text name=username>");
    out.print("<input type=submit value=提交>");
    out.print("</td></tr></form>");
    out.print("<TR><td colspan=8 align=center>用户数据
        (共"+rowNumber+"条记录)</td></tr>");
    out.print("<TR>");
        out.print("<Td width=100 >"+ "用户 ID 号</td>");
        out.print("<Td width=50 >"+ "用户名</td>");
        out.print("<Td width=100>"+ "用户真实姓名</td>");
        out.print("<Td width=50>"+ "年龄</td>");
        out.print("<Td width=50>"+ "性别</td>");
        out.print("<Td width=100>"+ "联系地址</td>");
        out.print("<Td width=100>"+ "联系电话</td>");
        out.print("<Td width=100>"+ "添加时间</td>");
    out.print("</TR>");
    rs.beforeFirst();
    while(rs.next()){
        out.print("<TR>");
        out.print("<TD >"+rs.getLong(1)+"</TD>");
        out.print("<TD >"+rs.getString(2)+"</TD>");
        out.print("<TD >"+rs.getString(4)+"</TD>");
        out.print("<TD >"+rs.getInt("user_age")+"</TD>");
        out.print("<TD >"+rs.getString("user_sex")+"</TD>");
        out.print("<TD >"+rs.getString("user_address")+"</TD>");
        out.print("<TD >"+rs.getString("user_telephone")+"</TD>");
    }
} catch (SQLException e) {
    out.print("<div>数据库连接失败</div>");
}
%>

```

```

        out.print("<TD>"+rs.getString("add_time")+"</TD>");
        out.print("</TR>");
    }
    out.print("</Table>");
    con.close();
} catch (SQLException e1) {
    out.print("SQL 异常!");
}
%></BODY>
</HTML>

```

程序实现了用户姓名的模糊查询，实际上与参数查询不同的是在构造 SQL 语句时，使用 where 的 like 来确定条件，即 user\_true\_name like '%username%'。程序的运行结果如图 6-13 所示（查询出了姓邓的用户信息）。



图 6-13 模糊查询

### 6.3.5 综合查询

综合查询就是要同时满足用户多个条件的查询，其程序实例如下。

#### 【实例 6-5】综合查询

selectUserTable6.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<%!
public String codeToString(String str){//处理中文字符串的函数
    String s=str;
    try{
        byte tempB[]=s.getBytes("ISO-8859-1");
        s=new String(tempB);
        return s;
    } catch (Exception e){
        return s;
    }
}
%>
<!--构造查询 SQL 语句
    String sqlString=null;//SQL 语句

```



```

String sex=codeToString(request.getParameter("sex"));
String username=codeToString(request.getParameter("username"));
if(sex==null||sex.trim().length()==0)
    sqlString=new String("SELECT * FROM userTable");
else
    sqlString=new String("SELECT * FROM userTable where user_sex='"+sex.trim()+"'");
if(username==null||username.trim().length()==0)//SQL 语句不变
    ;//此句什么事也不做
else
    if(sqlString.indexOf("where")==-1)//SQL 语句中没有 where 子句
        sqlString=sqlString+" where user_true_name like '%"+username+"%'";
    else
        sqlString=sqlString+" and user_true_name like '%"+username+"%'";
%>
<HTML>
<BODY>
<% Connection con;
Statement sql;
ResultSet rs;
try {
    con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
        ":1433;DatabaseName=testDatabase","sa","123");
    sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
    rs=sql.executeQuery(sqlString);
    rs.last();
    int rowNum=rs.getRow();
    out.print("<Table Border>");
    out.print("<form action=selectUserTable6.jsp method=post>");
    out.print("<TR><td>查询条件:</td>");
    out.print("<td colspan=7>");
    out.print("姓名:<input type=text name=username>");
    out.print("&nbsp;性别:<select name=sex>");
    out.print("<option value='' selected>所有</option>");
    out.print("<option value=男>男</option>");
    out.print("<option value=女>女</option></select>");
    out.print("<input type=submit value=提交>");
    out.print("</td></tr></form>");
    out.print("<TR><td colspan=8 align=center>用户数据
(共"+rowNum+"条记录)</td></tr>");
    out.print("<TR>");
    out.print("<Td width=100 >"+用户 ID 号</td>");
    out.print("<Td width=50 >"+用户名</td>");
    out.print("<Td width=100>"+用户真实姓名</td>");
    out.print("<Td width=50>"+年龄</td>");
    out.print("<Td width=50>"+性别</td>");
    out.print("<Td width=100>"+联系地址</td>");
    out.print("<Td width=100>"+联系电话</td>");
    out.print("<Td width=100>"+添加时间</td>");
    out.print("</TR>");
    rs.beforeFirst();
    while(rs.next())

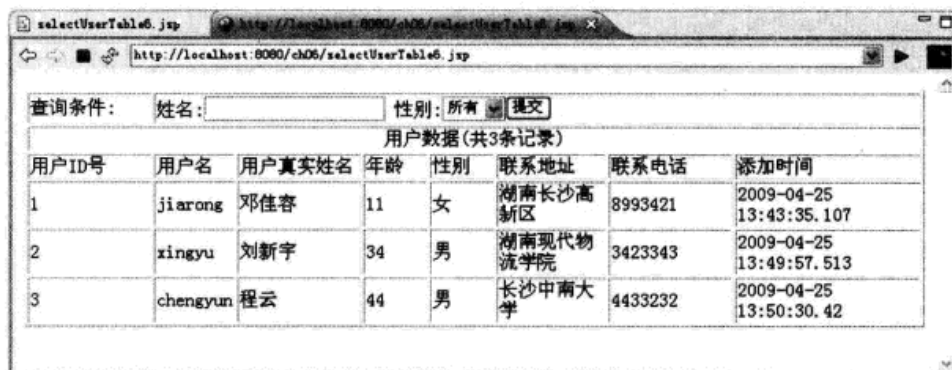
```

```

        { out.print("<TR>");
          out.print("<TD >"+rs.getLong(1)+"</TD>");
          out.print("<TD >"+rs.getString(2)+"</TD>");
          out.print("<TD >"+rs.getString(4)+"</TD>");
          out.print("<TD >"+rs.getInt("user_age")+"</TD>");
          out.print("<TD >"+rs.getString("user_sex")+"</TD>");
          out.print("<TD >"+rs.getString("user_address")+"</TD>");
          out.print("<TD >"+rs.getString("user_telephone")+"</TD>");
          out.print("<TD >"+rs.getString("add_time")+"</TD>");
          out.print("</TR>");
        }
        out.print("</Table>");
        con.close();
      }catch(SQLException e1) {
        out.print(e1);
      }
    }
  }>
</BODY>
</HTML>

```

该程序实现了同时查询姓名和性别。与前面所学的参数查询和模糊查询最大的区别就是在构造 SQL 语句处。这里应用了 String 类的 IndexOf() 方法, 这个方法是用来在字符串中查找子串, 如果存在指定的子串则返回子串所在位置, 如果不存在返回-1。因此可以用来查找 SQL 语句中是否含有 where 子句, 如果没有则在 SQL 语句中增加条件时要加入 where, 如果有则加入 and。程序的运行结果如图 6-14 所示。



查询条件: 姓名:  性别: 所有

用户数据(共3条记录)

用户ID号	用户名	用户真实姓名	年龄	性别	联系地址	联系电话	添加时间
1	jiarong	邓佳蓉	11	女	湖南长沙高新区	8993421	2009-04-25 13:43:35.107
2	xingyu	刘新宇	34	男	湖南现代物流学院	3423343	2009-04-25 13:49:57.513
3	chengyun	程云	44	男	长沙中南大学	4433232	2009-04-25 13:50:30.42

图 6-14 综合查询

实际应用中大多是多表的联合查询和更多个条件的综合查询, 只要在程序中正确地构造 SQL 语句, 实现其相应的查询功能并不难。

## 6.4 追加记录

追加记录就是在数据库中表的最后一行之后追加记录。其程序实例如下。

### 【实例 6-6】追加记录

该程序实现了向数据库中的表 userTable, 即用户信息表追加用户记录, 也就是注册用户。



## insertUserTable.jsp

```

<%@ page contentType="text/html;charset=GB2312" %>
<script language="javascript">
function on_submit(){//验证数据的合法性
    if (form1.username.value == ""){
        alert("用户名不能为空, 请输入用户名!");
        form1.username.focus();
        return false;
    }
    if (form1.password.value == ""){
        alert("用户密码不能为空, 请输入密码!");
        form1.password.focus();
        return false;
    }
    if (form1.repassword.value == ""){
        alert("用户确认密码不能为空, 请输入密码!");
        form1.repassword.focus();
        return false;
    }
    if (form1.password.value != form1.repassword.value){
        alert("密码与确认密码不同");
        form1.password.focus();
        return false;
    }
}
</script>
<head>
<title>用户注册程序</title>
</head>
<body>
<center>
<table border="1" width="700">
    <tr>
        <td width="100%" colspan="2" align="center">用户注册程序</td>
    </tr>
    <form action="acceptInsertUserTable.jsp" method="post"
        onsubmit="return on_submit()" name="form1">
    <tr>
        <td width="25%">用户名:</td>
        <td width="75%"><input type="text" name="username">(*)必填</td>
    </tr>
    <tr>
        <td>密码:</td>
        <td><input type="password" name="password">(*)必填</td>
    </tr>
    <tr>
        <td>再输入一次密码:</td>
        <td><input type="password" name="repassword">(*)必填</td>
    </tr>
    <tr>
        <td>真实姓名:</td>
        <td><input type="text" name="usertruenam"></td>
    </tr>

```

```

<tr>
  <td>年龄:</td>
  <td><input type="text" name="age"></td>
</tr>
<tr>
  <td>性别:</td>
  <td><input type="radio" name="sex" value="男" checked>男<input type="radio"
    value="女" name="sex">女</td>
</tr>
<tr>
  <td>联系地址:</td>
  <td><input type="text" name="address" size="60"></td>
</tr>
<tr>
  <td>联系电话:</td>
  <td><input type="text" name="telephone"></td>
</tr>
<tr>
  <td colspan="2" align="center"> <input type="submit" value="提交">
    <input type="reset" value="重输"></td>
</tr>
</form>
</table>
</center>
</body>
</html>

```

这个 JSP 页面实质上是一个输入用户数据的表单，用 JavaScript 进行数据验证，验证合法后再把数据提交给 `acceptInsertUserTable.jsp` 页面。本页面的运行结果如图 6-15 所示。

图 6-15 注册程序表单页面

#### `acceptInsertUserTable.jsp`

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<%!
public String codeToString(String str){//处理中文字符串的函数
  String s=str;
  try{

```



```

        byte tempB[]=s.getBytes("ISO-8859-1");
        s=new String(tempB);
        return s;
    }catch(Exception e){
        return s;
    }
}
%>
<%//接收客户端提交的数据
String username=codeToString(request.getParameter("username"));
if(username==null)//无内容则设为空串
    username="";
String password=codeToString(request.getParameter("password"));
if(password==null)//无内容则设为空串
    password="";
String usertrueName=codeToString(request.getParameter("usertrueName"));
if(usertrueName==null)//无内容则设为空串
    usertrueName="";
String age=codeToString(request.getParameter("age"));
int ageint;
try{
    ageint=Integer.parseInt(age.trim());
}catch(Exception e){
    ageint=0; //没有输入年龄或输入的年龄不是数字则值为 0
}
String sex=codeToString(request.getParameter("sex"));
if(sex==null)//无内容则设为空串
    sex="";
String address=codeToString(request.getParameter("address"));
if(address==null)//无内容则设为空串
    address="";
String telephone=codeToString(request.getParameter("telephone"));
if(telephone==null)//无内容则设为空串
    telephone="";
%>
<%//构造追加记录 SQL 语句
String sqlString=null;//SQL 语句
sqlString="insert into userTable (user_name, user_password, user_true_name, "+
    "user_age, user_sex, user_address,user_telephone)"+
    " values ('"+username+"', '"+password+"', '"+usertrueName+"', "+
    ageint+ ", '"+sex+"', '"+address+"', '"+telephone+"')";
%>
<%//执行 SQL 语句
try {
    Connection con;
    Statement sql;
    con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
        ":1433;DatabaseName=testDatabase","sa","123");
    sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
    sql.executeUpdate(sqlString);
    con.close();
}
catch(SQLException e1)

```



```

    {
        out.print("SQL 异常! ");
    }
%>
<head>
<title>用户注册程序</title>
</head>
<body>
<center>
<table border="1" width="700">
    <tr>
        <td width="100%" colspan="2" align="center">用户注册程序</td>
    </tr>
    <tr>
        <td width="100%" colspan="2">追加用户成功! </td>
    </tr>
</table>
</center>
</body>
</html>

```

接收用户提交的数据后, 要用这些数据来构造正确的“insert into”作为数据插入的 SQL 语句, 以向表中追加记录。处理接收的数据, 一是要注意对象为 null 时的处理, 字符串可以把它改为长度为 0 的空串; 二是要注意数值型数据的处理, 如果提交的数据不是数值型的, 在进行数值转换时就会产生异常, 这时可在程序中捕获异常, 把这个数据设为一个指定的默认值或是报错处理, 运行结果如图 6-16 所示。

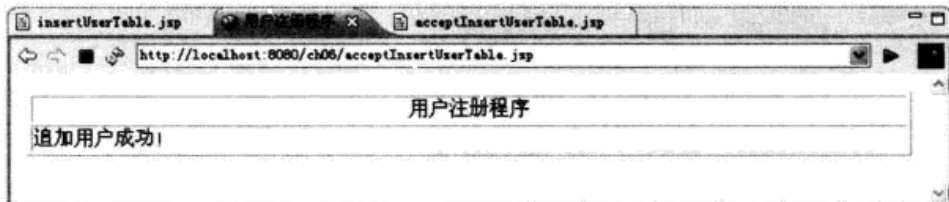


图 6-16 显示追加成功

## 6.5 删除记录

### 【实例 6-7】删除记录

下面的程序在其显示所有用户信息的页面中加入了一列, 显示出一个“删除”符号, 在“删除”符号上制作了一个超链接, 通过超链接传递用户 ID, 因为用户 ID 是主键, 可以唯一标识用户信息记录, 从而实现删除用户的信息。

deleteUser1.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<HTML><BODY>
    <% Connection con;

```



```

Statement sql;
ResultSet rs;
try {
    con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
        ":1433;DatabaseName=testDatabase","sa","123");
    sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
    rs=sql.executeQuery("SELECT * FROM userTable");
    out.print("<Table Border style='font-size: 10pt'>");
    out.print("<TR><td colspan=9 align=center>用户数据</td></tr>");
    out.print("<TR>");
        out.print("<Td width=100 >"+ "用户 ID 号</td>");
        out.print("<Td width=50 >"+ "用户名</td>");
        out.print("<Td width=100>"+ "用户真实姓名</td>");
        out.print("<Td width=50>"+ "年龄</td>");
        out.print("<Td width=50>"+ "性别</td>");
        out.print("<Td width=100>"+ "联系地址</td>");
        out.print("<Td width=100>"+ "联系电话</td>");
        out.print("<Td width=100>"+ "添加时间</td>");
        out.print("<Td width=50>"+ "删除</td>");
    out.print("</TR>");
    while(rs.next()){
        out.print("<TR>");
        out.print("<TD >"+rs.getLong(1)+"</TD>");
        out.print("<TD >"+rs.getString(2)+"</TD>");
        out.print("<TD >"+rs.getString(4)+"</TD>");
        out.print("<TD >"+rs.getInt("user_age")+"</TD>");
        out.print("<TD >"+rs.getString("user_sex")+"</TD>");
        out.print("<TD >"+rs.getString("user_address")+"</TD>");
        out.print("<TD >"+rs.getString("user_telephone")+"</TD>");
        out.print("<TD >"+rs.getString("add_time")+"</TD>");
        out.print("<TD ><a href='deleteUser2.jsp?user_id="
            "+rs.getLong(1)+'>x</a></TD>");
        out.print("</TR>");
    }
    out.print("</Table>");
    con.close();
} catch(SQLException e1){
    out.print(e1);
}
%>
</BODY></HTML>

```

页面的运行结果如图 6-17 所示。

单击用户信息后的“删除”超链接，就可删除相应的用户信息记录。删除记录 JSP 页面的程序代码如下所示。

#### deleteUser2.jsp

```


<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<%//接收要删除的用户 ID 号
long user_id;
try{

```

```

        user_id=Long.parseLong(request.getParameter("user_id"));
    }catch(Exception e){
        user_id=0;
    }
}

```



用户ID号	用户名	用户真实姓名	年龄	性别	联系地址	联系电话	添加时间	删除
1	jiarong	邓佳容	11	女	湖南长沙高新区	8993421	2009-04-25 13:43:35.107	<a href="#">×</a>
2	xingyu	刘新宇	34	男	湖南现代物流学院	3423343	2009-04-25 13:49:57.513	<a href="#">×</a>
3	chengyun	程云	44	男	长沙中南大学	4433232	2009-04-25 13:50:30.42	<a href="#">×</a>
4	xiaoming	刘小敏	34	女	首都师范大学	010-22332332	2009-04-25 14:21:34.077	<a href="#">×</a>

图 6-17 用户信息显示页面

```

%>
<%//构造删除记录 SQL 语句
String sqlString=null;//SQL 语句
if(user_id!=0) { //接收到的参数正确
    sqlString="delete from userTable where user_id="+user_id;
    //执行 SQL 语句
    try{
        Connection con;
        Statement sql;
        con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
            ":1433;DatabaseName=testDatabase","sa","123");
        sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        sql.executeUpdate(sqlString);
        con.close();
    } catch(SQLException e1) {
        out.print(e1);
    }
}
%>
<html>
<head>
<title>用户删除程序</title>
</head>
<body>
<center>
<table border="1" width="700">
    <tr>
        <td width="100%" colspan="2" align="center">删除用户程序</td>
    </tr>
    <tr>
        <td width="100%" colspan="2">删除用户成功! </td>
    </tr>
</table>
</center>
</body>
</html>

```



该程序中用 Long 包装类的 `parseLong()` 方法把接收到的用户 ID 转换为 long 型数据，再用这个数据来构造删除用户记录的 SQL 语句，然后执行 SQL 语句。程序的执行结果如图 6-18 所示。

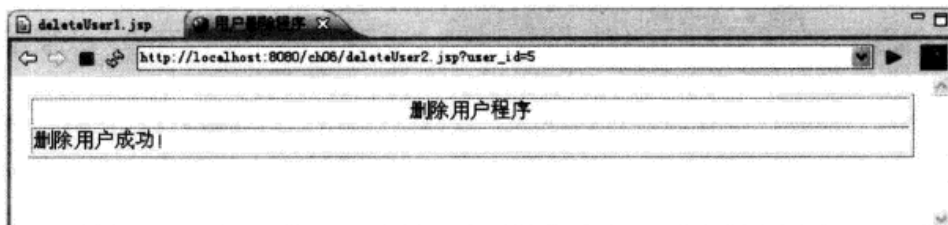


图 6-18 删除用户成功

## 6.6 更新记录

### 【实例 6-8】更新记录

这个程序实例演示了如何修改用户资料。第一个页面还是在实例 6-1 的基础上，在其显示所有用户信息的页面中加入了一列，显示出“修改资料”文字，在“修改资料”文字上制作了一个超链接，通过超链接传递用户 ID，因为用户 ID 是主键，可以唯一标识用户信息记录，从而实现修改用户的资料。

#### updateUser1.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<HTML>
<BODY>
<% Connection con;
    Statement sql;
    ResultSet rs;
    try {
        con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
            ":1433;DatabaseName=testDatabase","sa","123");
        sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        rs=sql.executeQuery("SELECT * FROM userTable");
        out.print("<Table Border style='font-size: 10pt'>");
        out.print("<TR><td colspan=9 align=center>用户数据</td></tr>");
        out.print("<TR>");
        out.print("<Td width=60 >"+ "用户 ID 号</td>");
        out.print("<Td width=50 >"+ "用户名</td>");
        out.print("<Td width=100>"+ "用户真实姓名</td>");
        out.print("<Td width=40>"+ "年龄</td>");
        out.print("<Td width=40>"+ "性别</td>");
        out.print("<Td width=100>"+ "联系地址</td>");
        out.print("<Td width=100>"+ "联系电话</td>");
        out.print("<Td width=100>"+ "添加时间</td>");
        out.print("<Td width=80>"+ "修改资料</td>");
```

```

        out.print("</TR>");
        while(rs.next()){
            out.print("<TR>");
            out.print("<TD >"+rs.getLong(1)+"</TD>");
            out.print("<TD >"+rs.getString(2)+"</TD>");
            out.print("<TD >"+rs.getString(4)+"</TD>");
            out.print("<TD >"+rs.getInt("user_age")+"</TD>");
            out.print("<TD >"+rs.getString("user_sex")+"</TD>");
            out.print("<TD >"+rs.getString("user_address")+"</TD>");
            out.print("<TD >"+rs.getString("user_telephone")+"</TD>");
            out.print("<TD >"+rs.getString("add_time")+"</TD>");
            out.print("<TD ><a href='updateUser2.jsp?user_id="
                +rs.getLong(1)+"'>修改资料</a></TD>");
            out.print("</TR>");
        }
        out.print("</Table>");
        con.close();
    }catch(SQLException e1) {
        out.print(e1);
    }
}
%>
</BODY>
</HTML>

```

页面的运行结果如图 6-19 所示。

用户ID号	用户名	用户真实姓名	年龄	性别	联系地址	联系电话	添加时间	修改资料
1	jiarong	邓佳容	11	女	湖南长沙高新区	8993421	2009-04-25 13:43:35.107	<a href="#">修改资料</a>
2	xingyu	刘新宇	34	男	湖南现代物流学院	3423343	2009-04-25 13:49:57.513	<a href="#">修改资料</a>
3	chengyun	程云	44	男	长沙中南大学	4433232	2009-04-25 13:50:30.42	<a href="#">修改资料</a>
4	xiaoming	刘小敏	34	女	首都师范大学	010-22332332	2009-04-25 14:21:34.077	<a href="#">修改资料</a>

图 6-19 修改资料显示页面

单击“修改资料”超链接，会传递用户 ID 参数，转到相应的用户资料编辑页面。下面的页面程序是用户资料编辑页面程序。

#### updateUser2.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<html>
<script language="javascript">
    function on_submit(){//验证数据的合法性
        if (form1.username.value == ""){
            alert("用户名不能为空，请输入用户名！");
            form1.username.focus();
            return false;
        }
    }

```



```

    }
}
</script>
<%//接收要修改的用户 ID 号
long user_id;
try{
    user_id=Long.parseLong(request.getParameter("user_id"));
}catch(Exception e){
    user_id=0;
}
%>
<head>
<title>修改用户资料程序</title>
</head>
<body>
<center>
<%if(user_id!=0){
    String sqlString="select * from userTable where user_id="+user_id;
    try {
        Connection con;
        Statement sql;
        ResultSet rs;
        con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
            ":1433;DatabaseName=testDatabase","sa","123");
        sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        rs=sql.executeQuery(sqlString);
        rs.next();
    }
%>
<table border="1" width="700">
    <tr>
        <td width="100%" colspan="2" align="center">修改用户资料程序</td>
    </tr>
    <form action="updateUser3.jsp" method="post"
        onsubmit="return on_submit()" name="form1">
    <input type="hidden" value="<%=user_id%>" name="user_id">
    <tr>
        <td width="25%">用户名:</td>
        <td width="75%"><input type="text" name="username"
            value="<%=rs.getString("user_name")%>"></td>
    </tr>
    <tr>
        <td>真实姓名:</td>
        <td><input type="text" name="usertruename"
            value="<%=rs.getString("user_true_name")%>"></td>
    </tr>
    <tr>
        <td>年龄:</td>
        <td><input type="text" name="age" value="<%=rs.getString("user_age")%>"> </td>
    </tr>
    <tr>
        <td>性别:</td>
        <td>

```



```

<%if((rs.getString("user_sex")).equals("男")){%>
<input type="radio" name="sex" value="男" checked>
<%}else{%>
<input type="radio" name="sex" value="男">
<%}%>
男
<%if((rs.getString("user_sex")).equals("女")){%>
<input type="radio" value="女" name="sex" checked>
<%}else{%>
<input type="radio" name="sex" value="女">
<%}%>
女
</td>
</tr>
<tr>
<td>联系地址:</td>
<td ><input type="text" name="address" size="60"
value="<%=rs. getString("user_address") %>"></td>
</tr>
<tr>
<td>联系电话:</td>
<td ><input type="text" name="telephone"
value="<%=rs.getString("user_telephone") %>"></td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="提交">
<input type="reset" value="重输"></td>
</tr>
</form>
</table>
<%
con.close();
}catch(SQLException e1) {
out.print(e1);
}
}else{
out.print("数据不正确!");
}%>
</center></body></html>

```

本程序中,使用了<%= %>的方式,在 HTML 标签中加入了从相应的数据库中取到的字段值;此外,程序中使用了 HTML 标签中的 hidden 输入标签,即隐藏标签,使用户 ID 不显示出来也可以提交。编辑完用户资料后把数据提交给 updateUser3.jsp 页面。如前面页面中选择了用户 ID 为 4 的用户,则运行结果如图 6-20 所示,单击“提交”按钮就会提交数据。

#### updateUser3.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<%!
public String codeToString(String str){//处理中文字符串的函数
String s=str;

```



```

try{
    byte tempB[]=s.getBytes("ISO-8859-1");
    s=new String(tempB);
    return s;
}catch(Exception e){
    return s;
}
}

```

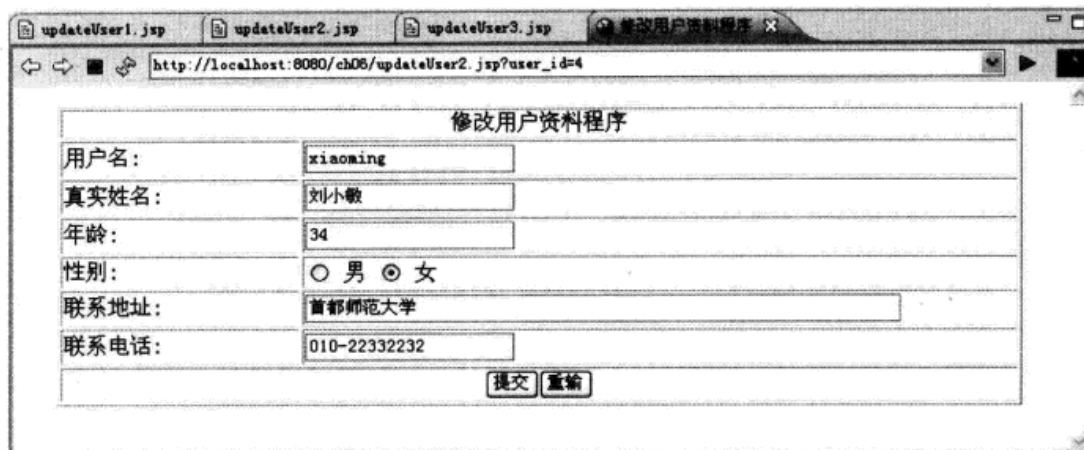


图 6-20 编辑用户资料页面

```

%>
<%//接收要修改的用户 ID 号
    long user_id=Long.parseLong(request.getParameter("user_id"));
%>
%>
<%//接收客户端提交的数据
    String username=codeToString(request.getParameter("username"));
    if(username==null)//无内容则设为空串
        username="";
    String usertruename=codeToString(request.getParameter("usertruename"));
    if(usertruename==null)//无内容则设为空串
        usertruename="";
    String age=codeToString(request.getParameter("age"));
    int ageint;
    try{
        ageint=Integer.parseInt(age.trim());
    }catch(Exception e){
        ageint=0; //没有输入年龄或输入的年龄不是数字则值为 0
    }
    String sex=codeToString(request.getParameter("sex"));
    if(sex==null)//无内容则设为空串
        sex="";
    String address=codeToString(request.getParameter("address"));
    if(address==null)//无内容则设为空串
        address="";
    String telephone=codeToString(request.getParameter("telephone"));
    if(telephone==null)//无内容则设为空串
        telephone="";
%>

```



```

<%//构造修改记录 SQL 语句
String sqlString=null;//SQL 语句
sqlString="update userTable set user_name='"+
    username+"',user_true_name= '"+usertruenam+"',user_age="+
    ageint+", user_sex='"+sex+"', user_address='"+address+"', user_telephone='"+
    telephone+"' where user_id="+user_id;
%>
<%//执行 SQL 语句
try {
    Connection con;
    Statement sql;
    con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
        ":1433;DatabaseName=testDatabase","sa","123");
    sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
    sql.executeUpdate(sqlString);
    con.close();
}catch(SQLException e1) {
    out.print(e1);
}
%>
<head>
<title>修改用户资料程序</title>
</head>
<body>
<center>
<table border="1" width="700">
    <tr>
        <td width="100%" colspan="2" align="center">修改用户资料程序</td>
    </tr>
    <tr>
        <td width="100%" colspan="2">修改用户资料成功! </td>
    </tr>
</table>
</center>
</body>
</html>

```

这里其实最重要的还是构造 SQL 语句,这里构造的是“update”这一更新数据库记录的 SQL 语句,构造时要注意符号“'”的配对使用,程序的运行结果如图 6-21 所示。

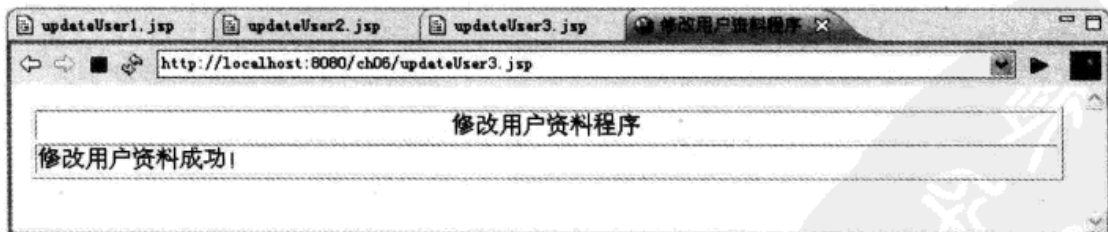


图 6-21 修改用户资料成功



## 6.7 在 ResultSet 中修改数据

### 6.7.1 追加记录

在 6.4、6.5、6.6 节中使用的都是通过构造 SQL 语句中的 insert 语句实现追加记录功能，用 delete 语句实现删除记录功能，用 update 语句实现更新记录数据功能。能不能不需要构造这些 SQL 语句就可以修改记录数据呢？当然可以，用 ResultSet 类就可以实现。首先查询记录的语句必须是可更新的，即连接数据库对象创建语句 createStatement() 方法中的第二个参数值是 ResultSet.CONCUR\_UPDATABLE；然后再调用 ResultSet 的相应方法。

在实例 6-6 追加记录的程序实例中，可将 acceptInsertUserTable.jsp 中的代码做如下修改。

```
<!--构造追加记录 SQL 语句
String sqlString=null;//SQL 语句
sqlString = "insert into userTable (user_name, user_password, "+
    "user_true_name,user_age,user_sex,user_address,user_telephone)"+
    "values('"+username+"','"+password+"','"+usertrueName+"','"+ageint+"','"+
    sex+"','"+address+"','"+telephone+"')";
%>
<!--执行 SQL 语句
try {
    Connection con;
    Statement sql;
    con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
        ":1433;DatabaseName=testDatabase","sa","123");
    sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
    sql.executeUpdate(sqlString);
    con.close();
} catch(SQLException e1) {
    out.print(e1);
}
%>
```

把这两段 Java 代码更改为如下的一段代码。

```
<!--用 ResultSet 追加记录
String sqlString=null;//SQL 语句
try {
    Connection con;
    Statement sql;
    ResultSet rs;
    con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
        ":1433;DatabaseName=testDatabase","sa","123");
    sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
    sqlString="select * from userTable";
    rs=sql.executeQuery(sqlString);
    rs.moveToInsertRow();
    resultset.updateString("user_name",username);
    resultset.updateString("user_password",password);
```

```

        resultSet.updateString("user_true_name",usertruename);
        resultSet.updateInt("user_age",ageint);
        resultSet.updateString("user_sex",sex);
        resultSet.updateString("user_address",address);
        resultSet.updateString("user_telephone",telephone);
        rs.insertRow();
        con.close();
    }catch(SQLException e1){
        out.print(e1);
    }
}
%>

```

同样实现了追加记录的功能。在用 **ResultSet** 追加记录的代码段中, 首先创建一个支持通过结果集向表插入数据的 **ResultSet** 对象, 然后调用 **ResultSet** 对象的 **moveToInsertRow()** 方法移动游标, 再调用 **ResultSet** 结果集的 **insertRow()** 方法, 将数据写入到数据库中。



**注意** **ResultSet** 对象调用 **moveToInsertRow()** 后, 记录指针指向的不是一条记录而是内存中的一块缓冲区, 在这个缓冲区中根据实际数据库中表的特征为每一个字段开辟一个相应的区域, 在其后的 **update × × ×()** 方法数据就写在这个缓冲区中。

## 6.7.2 删除记录

在实例 6-7 的删除记录程序实例中, 把 **deleteUser2.jsp** 中的代码修改为如下:

```

<%//构造删除记录 SQL 语句
String sqlString=null;//SQL 语句
if(user_id!=0) { //接收到的参数正确
    sqlString="delete from userTable where user_id="+user_id;
    //执行 SQL 语句
    try {
        Connection con;
        Statement sql;
        con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
            ":1433;DatabaseName=testDatabase","sa","123");
        sql=con.createStatement();
        sql.executeUpdate(sqlString);
        con.close();
    } catch(SQLException e1) {
        out.print(e1);
    }
}
}
%>

```

把这段 Java 代码更改为如下代码。

```

<%
String sqlString=null;//SQL 语句
if(user_id!=0) { //接收到的参数正确
    sqlString="select * from userTable where user_id="+user_id;
    //执行 SQL 语句
    try {
        Connection con;

```



```

        Statement sql;
        ResultSet rs;
        con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
            ":1433;DatabaseName=testDatabase","sa","123");
        sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
        rs=sql.executeQuery(sqlString);
        rs.next();
        rs.deleteRow();
        con.close();
    }catch(SQLException e1) {
        out.print(e1);
    }
}
%>

```

该程序中首先定位记录指针到要删除的记录，再调用 `deleteRow()` 方法删除记录。

### 6.7.3 更新记录

在实例 6-8 的更新记录程序实例中，对 `updateUser3.jsp` 代码作如下修改。

```

<%//构造修改记录 SQL 语句
String sqlString=null;//SQL 语句
sqlString="update userTable set user_name='"+username+
    "','user_true_name='"+usertruename+"',user_age="+ageint+
    "','user_sex='"+sex+"',user_address='"+address+"',user_telephone='"+
    telephone+"' where user_id="+user_id;
%>
<%//执行 SQL 语句
try {
    Connection con;
    Statement sql;
    con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
        ":1433;DatabaseName=testDatabase","sa","123");
    sql=con.createStatement();
    sql.executeUpdate(sqlString);
    con.close();
}catch(SQLException e1) {
    out.print(e1);
}
%>

```

把这两段代码更改为一段。

```

<%
String sqlString=null;//SQL 语句
if(user_id!=0){//接收到的参数正确
    sqlString="select * from userTable where user_id="+user_id;
    //执行 SQL 语句
    try {
        Connection con;
        Statement sql;
        ResultSet rs;

```

```

        con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
            ":1433;DatabaseName=testDatabase","sa","123");
        sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
        rs=sql.executeQuery(sqlString);
        rs.next();
        resultset.updateString("user_name",username);
        resultset.updateString("user_true_name",usertruename);
        resultset.updateInt("user_age",ageint);
        resultset.updateString("user_sex",sex);
        resultset.updateString("user_address",address);
        resultset.updateString("user_telephone",telephone);
        rs.updateRow();
        con.close();
    }catch(SQLException e1) {
        out.print(e1);
    }
}
%>

```

该程序中调用 `updateRow()` 方法更改数据库中的数据。`UpdateXXX()` 方法不会更改实际数据库中的数据,只是更改缓冲区中的数据,如果要撤销修改可调用 `cancelRowUpdates()` 方法取消变更。

## 6.8 分页显示记录

在 6.3 节中查询记录时,如果查询出来的数据记录条数较多,所有的数据在一个 JSP 页面中显示时页面会拉得很长,显示出来的效果并不美观,把数据分页显示出来是一种好的办法。

### 【实例 6-9】分页显示记录

userPage1.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<%
    int dipage=1; //当前页码数默认为 1
    String pages=request.getParameter("dipage");
    if(pages==null){
        pages="1";
    }
    try{
        dipage=Integer.parseInt(pages);
    }catch(Exception e){
        dipage=1;
    }
%>
<HTML>
<title>用户数据</title>
<BODY>
    <% Connection con;

```



```

Statement sql;
ResultSet rs;
try {
    con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
        ":1433;DatabaseName=testDatabase","sa","123");
    sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
    rs=sql.executeQuery("SELECT * FROM userTable");
    int countRecord=0;//记录条数
    int countPageRecord=0;//每页记录条数
    int countPage=0;//总页数
    countPageRecord=5;//每页 5 条记录, 要设置每页记录条数就更改这个变量的值
    //得到记录的条数
    rs.last();
    countRecord=rs.getRow();
    //得到总页数
    if(countRecord%countPageRecord==0)
        countPage=countRecord/countPageRecord;
    else
        countPage=countRecord/countPageRecord+1;
    //把记录指针移至当前页第一条记录之前
    if((dipage-1)*countPageRecord==0)
        rs.beforeFirst();
    else
        rs.absolute((dipage-1)*countPageRecord);
    out.print("<Table Border style='font-size: 10pt'>");
    out.print("<TR><td colspan=8 align=center>用户数据</td></tr>");
    out.print("<TR>");
        out.print("<Td width=60 >"+ "用户 ID 号</td>");
        out.print("<Td width=50 >"+ "用户名</td>");
        out.print("<Td width=100>"+ "用户真实姓名</td>");
        out.print("<Td width=40>"+ "年龄</td>");
        out.print("<Td width=40>"+ "性别</td>");
        out.print("<Td width=100>"+ "联系地址</td>");
        out.print("<Td width=100>"+ "联系电话</td>");
        out.print("<Td width=100>"+ "添加时间</td>");
    out.print("</TR>");
    int i=0;
    while(rs.next()){
        out.print("<TR>");
            out.print("<TD >"+rs.getLong(1)+"</TD>");
            out.print("<TD >"+rs.getString(2)+"</TD>");
            out.print("<TD >"+rs.getString(4)+"</TD>");
            out.print("<TD >"+rs.getInt("user_age")+"</TD>");
            out.print("<TD >"+rs.getString("user_sex")+"</TD>");
            out.print("<TD >"+rs.getString("user_address")+"</TD>");
            out.print("<TD >"+rs.getString("user_telephone")+"</TD>");
            out.print("<TD >"+rs.getString("add_time")+"</TD>");
        out.print("</TR>");
        i++;
        if(i>=countPageRecord) break; //当前页显示完, 则退出循环
    }
    out.print("<TR><td colspan=8 align=center>");

```

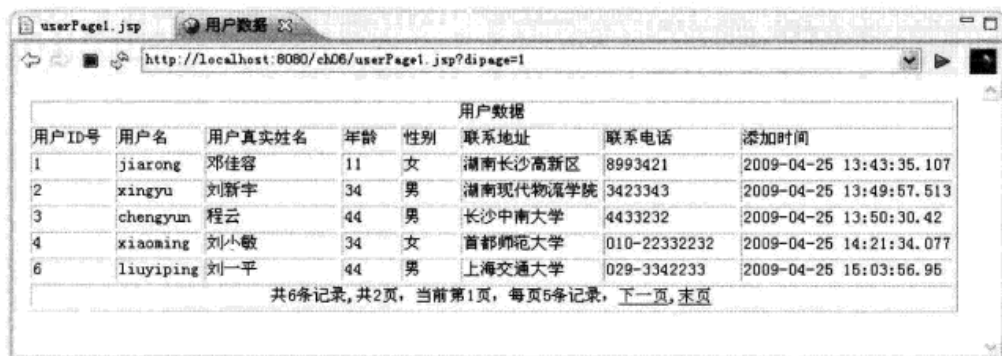


```

        out.print("共"+countRecord+"条记录,共"+countPage+"页,当前第"+dipage+
            "页,每页"+countPageRecord+"条记录,");
        if(dipage==1);//当前是首页
        else{//当前不是首页
            out.print("<a href=userPage1.jsp?dipage=1>首页</a>,");
            out.print("<a href=userPage1.jsp?dipage="+ (dipage-1) + ">上一页</a>,");
        }
        if(dipage==countPage) ;//当前是末页
        else{//当前不是末页
            out.print("<a href=userPage1.jsp?dipage="+ (dipage+1) + ">下一页</a>,");
            out.print("<a href=userPage1.jsp?dipage="+countPage+">末页</a>");
        }
        out.print("</td></tr>");
    }
    out.print("</Table>");
    con.close();
} catch (SQLException e1) {
    out.print(e1);
}
%>
</BODY>
</HTML>

```

程序中首先用 **dipage** 变量接收传过来的页码参数,因为程序中用超链接捎带页码参数,这里要区分第一次访问页面时和以超链接访问时的情况,所以把默认时页码数设为 1; 打开数据表后,得到记录的条数和总页数;通过 **absolute()** 方法定位数据库记录指针到当前页第一行之前,再逐行显示出来。程序的运行结果如图 6-22 所示,这是第二页。



用户ID号	用户名	用户真实姓名	年龄	性别	联系地址	联系电话	添加时间
1	jiarong	邓佳容	11	女	湖南长沙高新区	8993421	2009-04-25 13:43:35.107
2	xingyu	刘新宇	34	男	湖南现代物流学院	3423343	2009-04-25 13:49:57.513
3	chengyun	程云	44	男	长沙中南大学	4433232	2009-04-25 13:50:30.42
4	xiaoming	刘小敏	34	女	首都师范大学	010-22332232	2009-04-25 14:21:34.077
6	liuyiping	刘一平	44	男	上海交通大学	029-3342233	2009-04-25 15:03:56.95

共6条记录,共2页,当前第1页,每页5条记录,下一页,末页

图 6-22 用户数据的分页显示

本例所介绍的分页显示记录的方法是最为简单的一种方法,但是这种方法需要用 SQL 语句一次性将所有满足条件的记录查询出来,这样如果结果记录集比较大的话,会占用不少的网络带宽,能否只查询出当前页的数据呢?当然可以,关键就在于构造出只查询当前页数据的 SQL 语句。

构造查询当前页数据的 SQL 语句的程序代码如下。

```

String sqlStr="select top 10 * from userTable where user_id not in"+
    " (select top "+(dipage-1)*countPageRecord+" user_id from "+
    " userTable order by user_id asc) order by user_id asc";

```



查询出当前页记录的原理图如图 6-23 所示。SQL 语句中，select 子句的作用是查询出当前页之前的页的记录，再查询出前 10 条不在当前页之前的页中的记录即为当前页的记录。

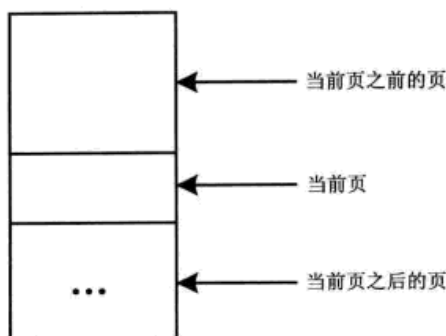


图 6-23 查询出当前页数据的原理图

为了加快程序的处理速度，还可以把 SQL 语句封装在数据库的存储过程中，再在 JSP 页面中调用，从而得到当前页的结果记录集，存储过程是预编译的，比临时执行 SQL 语句效率更高。

下面来看运用了以上思想的分页显示记录的程序代码。

#### userPage2.jsp

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.sql.*" %>
<%
int dipage=1;//当前页码数默认为 1
String pages=request.getParameter("dipage");
if(pages==null){
    pages="1";
}
try{
    dipage=Integer.parseInt(pages);
}catch(Exception e){
    dipage=1;
}
%>
<HTML>
<title>用户数据</title>
<BODY>
<% Connection con;
Statement sql;
ResultSet rs;
try {
    con=DriverManager.getConnection("jdbc:sqlserver://localhost+
":1433;DatabaseName=testDatabase","sa","123");
    sql=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
    rs=sql.executeQuery("SELECT count(*) as recordCount FROM userTable");
    int countRecord=0;//记录条数
    int countPageRecord=0;//每页记录条数
    int countPage=0;//总页数
    countPageRecord=5;//每页 5 条记录，要设置每页记录条数就更改这个变量的值
    //得到记录的条数
```



android与iphone及ipad开发书籍

-----持续不断更新中.....

C、C++、C#语言pdf书籍及vip视频教程

C、C++、C#、VC等-----持续不断更新中.....

delphi《书籍》及《视频》教程

-----持续不断更新中.....

E网管深VIP系列视频教程

黑客破解菜鸟修炼班，VB编程学习班，仿站学习培训，免杀培训，个人系统攻防系列教程，服务器搭建学习班，PHOTOSHOP平面设计班，基础制作论坛（论坛网站搭建），网赚系列教程，网站建设教程，网站漏洞基础，远程控制教程，软件破解班，脚本漏洞提权班

IT9网络学院VIP系列视频教程

免杀培训班，VMware虚拟机，零基础学习C语言，网游外挂开发精品系列语音教程（外挂教程学习必备研修31课全），VB语言教程30课全，Delphi编程到精通，远程控制软件，加解密解密，网络安全与黑客攻防培训，从入门到精通完整系统化学习C++编程，从入门到精通零基础学习汇编，wordpress教程(个人博客系统49课全)，外行人做易语言盗号和钓鱼程序语音教程

网址：WLSAM168.400GB.COM

Java书籍

-----持续不断更新中.....

photoshop、CorelDRAW、AutocAD等图像处理书籍及vip视频教程

-----持续不断更新中.....

powerbuilder书籍大全

Visual Basic语言vip视频教程及pdf书籍

-----持续不断更新中.....

windows、linux系统开发、系统封装等pdf书籍及VIP视频教程

-----持续不断更新中.....

《3DS Max》pdf书籍

《汇编语言》、《反汇编》及《调试》pdf书籍及vip视频教程

-----持续不断更新中.....

《电子书、电子书、还是电子书》pdf专题库

编程开发，家居美食，儿童益智，人物传记，增强记忆，快速阅读

信息系统项目管理师、网络工程师、系统分析师等软考类书籍

华中红客系列vip视频教程

脚本攻防培训班，源码免杀培训班，Css语言培训班，C语言，Dreamweaver网页设计，html网页设计培训班，PC安全班，php脚本语言培训班，VMWare虚拟机专题，webshell提权培训班，防站教程，零基础免杀培训班，刷钻速成班，脱壳破解班，外挂编写班，网络赚钱培训班，网站入侵培训班

外挂、驱动、逆向及封包视频教程

郁金香、独立团、夜猫论坛、天都吧、看流星论坛、一切从零开始等等

安全中国系列vip视频教程

易语言软件编程培训班，ASP.net网站开发项目实战培训班

我的收藏

按键精灵及TC脚本开发软件视频教程

-----持续不断更新中.....

当前位置： / 《电子书、电子书、还是电子书》pdf专题库 ←		
文件名	P D F电子书专题库，内容详尽，每天不断更新！！	
	办公类软件使用指南	
	医学	
	历史人物传记	
	哲学宗教	
	外语资料（除英语外）	（除英语外）
	官场类小说	本网盘内容太多，持续不断更新，发布各类视频教程、pdf书籍，包括破解、加解密、外挂辅助制作，易语言培训教程、编程语言、网页制作等等，教程及书籍仅用于学习，如用于商业或非法用途的后果自负！
	建筑工程类	
	情感生活类小说	
	政治军事	
	教育学习科普大全	网址：WLSAM168.400GB.COM
	文学理论	
	智力开发、增强记忆、快速阅读技巧大全	
	社会生活	
	科学技术	
	程序编程类	
	经济管理	
	网络安全及管理	
	网赚系列	
	美食小吃烹饪煲汤大全	
	课外读物	

































































































































































































































































































































[电脑世界的通关密语：电脑编程基础].(杉浦贤).滕永红.扫描版.pdf

[程序语言的奥妙：算法解读（四色全彩）].(杉浦贤).李克秋.扫描版.pdf

[差错：软件错误的致命影响].(帕伯斯).卞宇恒等.扫描版.pdf

[算法之道（第2版）].邹恒明.扫描版.pdf

[O'Reilly：深入学习MongoDB].(霍多罗夫).巨成等.扫描版.pdf

[深入浅出WPF].刘铁猛.扫描版.pdf

[Go语言-云动力（云计算时代的新型编程语言）].樊虹剑.扫描版.pdf

[精通.NET互操作：P/ Invoke、C++ Interop和COM Interop].黄际洲等.扫描版.pdf

[编程的奥秘：.NET软件技术学习与实践].金旭亮.扫描版.pdf

[O'Reilly：学习OpenCV（中文版）].(布拉德斯基等).于仕琪等.扫描版.pdf

[Go语言编程].许式伟等.扫描版.pdf

网址：WLSAM168.400GB.COM

[MySQL技术内幕：SQL编程].姜承尧.扫描版.pdf

[Tomcat权威指南（第2版）].(布里泰恩等).吴豪等.扫描版.pdf

[Ext江湖].大漠穷秋.扫描版.pdf

[IT名人堂-Oracle DBA突击：帮你赢得一份DBA职位].张晓明.扫描版.pdf

Total: 77

1

2

3

4

5

6

>

HTTP://WLSAM168.400GB.COM



```

if(rs.next())
    countRecord=rs.getInt("recordCount");
//得到总页数
if(countRecord%countPageRecord==0)
    countPage=countRecord/countPageRecord;
else
    countPage=countRecord/countPageRecord+1;
String sqlStr="select top 10 * from userTable where user_id not in"+
    " (select top "+(dipage-1)*countPageRecord+" user_id from "+
    " userTable order by user_id asc) order by user_id asc";
rs=sql.executeQuery(sqlStr);
out.print("<Table Border style='font-size: 10pt'>");
out.print("<TR><td colspan=8 align=center>用户数据</td></tr>");
out.print("<TR>");
    out.print("<Td width=60 >"+ "用户 ID 号</td>");
    out.print("<Td width=50 >"+ "用户名</td>");
    out.print("<Td width=100>"+ "用户真实姓名</td>");
    out.print("<Td width=40>"+ "年龄</td>");
    out.print("<Td width=40>"+ "性别</td>");
    out.print("<Td width=100>"+ "联系地址</td>");
    out.print("<Td width=100>"+ "联系电话</td>");
    out.print("<Td width=100>"+ "添加时间</td>");
out.print("</TR>");
int i=0;
while(rs.next()){
    out.print("<TR>");
    out.print("<TD >"+rs.getLong(1)+"</TD>");
    out.print("<TD >"+rs.getString(2)+"</TD>");
    out.print("<TD >"+rs.getString(4)+"</TD>");
    out.print("<TD >"+rs.getInt("user_age")+"</TD>");
    out.print("<TD >"+rs.getString("user_sex")+"</TD>");
    out.print("<TD >"+rs.getString("user_address")+"</TD>");
    out.print("<TD >"+rs.getString("user_telephone")+"</TD>");
    out.print("<TD >"+rs.getString("add_time")+"</TD>");
    out.print("</TR>");
    i++;
    if(i>=countPageRecord) break; //当前页显示完，则退出循环
}
out.print("<TR><td colspan=8 align=center>");
    out.print("共"+countRecord+"条记录,共"+countPage+"页,当前第"+dipage+
        "页,每页"+countPageRecord+"条记录,");
    if(dipage==1);//当前是首页
    else{//当前不是首页
        out.print("<a href=userPage2.jsp?dipage=1>首页</a>," );
        out.print("<a href=userPage2.jsp?dipage="+ (dipage-1) + ">上一页</a>," );
    }
    if(dipage==countPage) ;//当前是末页
    else{//当前不是末页
        out.print("<a href=userPage2.jsp?dipage="+ (dipage+1) + ">下一页</a>," );
        out.print("<a href=userPage2.jsp?dipage="+countPage+">末页</a>");
    }
}

```



```

        out.print("</td></tr>");
        out.print("</Table>");
        con.close();
    } catch (SQLException e1) {
        out.print(e1);
    }
}
%>
</BODY>
</HTML>

```

## 6.9 调用存储过程

调用存储过程对数据库进行操作比直接调用 SQL 语句具有更高的性能和效率。存储过程是存储在数据库管理系统服务器上的若干条经过预编译的 SQL 语句。存储过程可以输入也可以输出，支持用户设计的变量和流程控制，有强大的编程功能。

存储过程存储在数据库服务器上，由数据库管理系统管理控制，由服务器直接执行。一是因为存储过程是预编译的，因此相比执行 SQL 语句的方法节约了 SQL 语句的编译时间；二是可以减少网络传输数据量，这是由于它驻留在服务器上，不必等待记录通过网络传递过来就可以进行处理，而且如果显示的数据要作分页处理，可以有选择性的查询出部分数据，从而降低了数据的流量。正因为如此，在许多的系统中把一些针对数据库操作的应用逻辑甚至是业务逻辑都封装在了数据库的存储过程中。

在系统分析设计阶段，笔者也常常碰到这样的苦恼，差不多在每次的软件项目开发任务的技术会议上，都会有这样的讨论：到底是应当把逻辑封装在数据库中还是封装在应用程序中？每次技术人员的讨论都十分激烈。在此作一个小结，把逻辑封装在数据库中可以获得很高的效率但会使得逻辑变得混淆，一些应用逻辑和业务逻辑错综复杂地交互在一起；封装在应用程序中会使得需要快速地数据库访问的操作变得效率低下，特别是多表联合操作和保持数据库操作的多条 SQL 语句的原子性时。因此，在分析设计阶段要仔细弄清楚哪些逻辑应当封装在哪里，切不可一概而论。

在 Java 中要通过 CallableStatement 对象来访问数据库以调用存储过程，看下面的实例。

### 【实例 6-10】调用存储过程

首先在 SQL Server 的 testDatabase 数据库中创建如下的存储过程，创建语句如下。

```

CREATE PROCEDURE update_user_data
    (@user_name varchar(40),
     @user_true_name varchar(40),
     @user_age int,
     @user_sex varchar(2),
     @user_address varchar(80),
     @user_telephone varchar(20),
     @user_id bigint
    ) AS
    update userTable set user_name=@user_name,user_true_name=@user_true_name,
        user_age=@user_age,user_sex=@user_sex,user_address=@user_address,

```



```

        user_telephone=@user_telephone where user_id=@user_id
    select * from userTable where user_id=@user_id
GO

```

这个存储过程的功能是先更新记录再返回更新后的记录。

对 `updateUser3.jsp` 中构造 SQL 语句和执行 SQL 语句的代码段更改为如下的 Java 代码段。

```

<%//调用存储过程更新记录
String procedure_name="update_user_data";//执行的存储过程名称
String call_procedure_string="{call "+procedure_name+"(?,?,?,?,,?)}";
                                //调用存储过程的语句

try{
    Connection con;
    con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
        ":1433;DatabaseName=testDatabase","sa","123");
    CallableStatement callable_statement =
        con.prepareCall(call_procedure_string);//创建对象
    //设置调用存储过程的参数
    callable_statement.setString(1,username);
    callable_statement.setString(2,usertruename);
    callable_statement.setInt(3,ageint);
    callable_statement.setString(4,sex);
    callable_statement.setString(5,address);
    callable_statement.setString(6,telephone);
    callable_statement.setLong(7,user_id);
    callable_statement.executeUpdate();
    con.close();
}catch(SQLException e1) {
    out.print(e1);
}
%>

```

同样可以实现更新用户记录的功能。这里没有用 `ResultSet` 对象接收调用存储过程返回的记录集,实际应用中可以声明一个 `ResultSet` 对象来接收。程序中用 `Connection` 对象的 `prepareCall()` 方法把调用存储过程的 `CallableStatement` 对象与调用的语句关联起来,其中 `call` 语句的格式是 Java 中调用各种数据库中存储过程的通用格式,建立起关联后并没有执行调用,再用 `CallableStatement` 对象的 `setXxxx()` 方法设置调用语句中“?”外的参数值,设置完后用 `executeUpdate()` 方法调用存储过程。

## 6.10 事务处理

在对数据库的操作中,事务是一个独立的运行处理单元,它具有原子性,什么是原子性呢?它是指在一个事务中所有针对数据库的操作被封装起来,作为一个运行的单位,其中的若干个操作或者全都做或者全都不做。

Java 中用 `Connection` 对象的 `commit()` 方法提交事务,用 `rollback()` 方法回滚事务,回滚事务就是把事务中已做的操作取消,恢复到事务操作前数据库的状态。JDBC 中默认的方式是自动提交方式,用 `setAutoCommit()` 方法可以设定是否自动提交。这些方法的语法可以参见 6.2.5 节中的内容。



## 【实例 6-11】事务处理

transcation1.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<html>
<head>
<title>事务处理程序</title>
</head>
<body>
<center>
<table border="1" width="700">
<tr>
<td width="100%" colspan="2" align="center">事务处理程序</td>
</tr>
<tr>
<td width="100%" colspan="2">
<%//事务处理
String sqlString1="update userTable set user_age=30 where user_true_name like '邓%';
String sqlString2="update userTable set user_age=40 where user_true_name like '张%';
Connection con;
Statement sql;
con=DriverManager.getConnection("jdbc:sqlserver://localhost"+
":1433;DatabaseName=testDatabase","sa","123");
try { //设置不会自动提交
con.setAutoCommit(false);
sql=con.createStatement();
sql.executeUpdate(sqlString1);
sql.executeUpdate(sqlString2);
//提交事务
con.commit();
out.print("事务提交成功, 执行两条 SQL 语句。");
con.close();
}catch(SQLException e){ //操作不成功, 回滚事务
con.rollback();
out.print("事务操作不成功, 事务已回滚!");
}
%>
</td>
</tr>
</table>
</center>
</body>
</html>

```

在实践中,经常要用到事务处理,一个典型的实例是银行中的账务处理,采用复式记账法,两个账户在一笔交易中都需要操作,为保持一致性,必须应用事务来解决这个处理过程,要么两个账户操作都不成功,要么都成功。本例程序的运行结果如图 6-24 所示。



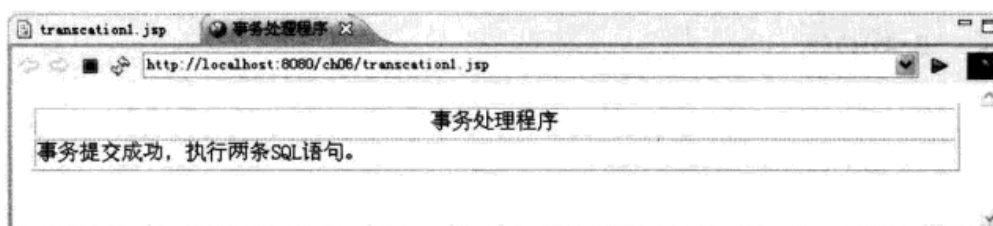


图 6-24 事务处理程序

## 6.11 连接其他数据库

下面来看用 JDBC 连接各种数据库的方式, 其实它们最大的区别就在于驱动的加载语句, 加载后会得到连接数据对象 `Connection`, 再通过 `Connection` 对象执行 SQL 语句, 之后的 Java 语句相同, 因此这里仅列出连接各种数据库不同的 Java 语句部分。

### 6.11.1 Oracle

```
<%
Connection conn= DriverManager.getConnection(
    "jdbc:Oracle:thin:@主机:端口号:数据库名","用户名","密码");
...
%>
```

### 6.11.2 MySQL

```
<%
Connection conn= DriverManager.getConnection(
    "jdbc:mysql://主机:端口号:数据库名","用户名","密码");
...
%>
```

### 6.11.3 Informix

```
<%
Connection conn= DriverManager.getConnection("jdbc:informix-sqli://主机:端口号
/数据库名:INFORMIXSERVER=informix 服务名","用户名","密码");
...
%>
```

### 6.11.4 Sybase

```
<%
Connection conn= DriverManager.getConnection(
    "jdbc:sybase:Tds:主机:端口号/数据库名","用户名","密码");
...
%>
```

### 6.11.5 AS400

```
<%
java.sql.DriverManager.registerDriver(new
com.ibm.as400.access.AS400JDBCDriver());
DriverManager.getConnection("jdbc:as400://主机","用户名","密码");
...
%>
```



**注意** 针对不同的数据库要在服务器上安装相应的 JDBC 驱动，驱动程序可从相应的数据库厂商的网站上去下载，最好参考一下数据库厂商提供的 JDBC 驱动文档，因为可能与以上的数据库连接语句会稍微有差别。

## 6.12 连接池技术

### 6.12.1 什么是 Connection Pool

数据库连接是一种重要的、有限的昂贵资源，当访问网站的用户越来越多时就显得尤为突出。这使得需要加强对数据库连接的管理，并显著影响到整个应用程序的伸缩性和健壮性，影响到程序的性能指标。于是出现了数据库连接池（Connection Pool），这项技术能明显提高对数据库操作的性能。

数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不再是重新建立一个；释放空闲时间超过最大空闲时间的数据库连接来避免因为没有释放数据库连接而引起的数据库连接遗漏。

### 6.12.2 Tomcat 7 上 Connection Pool 的配置

Tomcat 7 内置的数据库连接池为 DBCP（Database Connection Pool），DBCP 是 Jakarta Commons 的一个子项目。Tomcat 7 的 DBCP 组件包为 tomcat-dbcp.jar，位于 Tomcat 7 安装目录的 lib 子目录中。

修改 Tomcat 7 安装目录 conf 子目录下的 server.xml 文件，在</Host>之前加入如下配置：

```
<Resource name="jdbc/sqlserver" auth="Container" type="javax.sql.DataSource"
url="jdbc:sqlserver://localhost:1433;DatabaseName=testDatabase"
username="sa" password="123" maxActive="5000" maxIdle="10" maxWait="-1" />
```



**注意** 在配置连接池之前，如果使用 JDBC 方式请安装相应的 JDBC 驱动。且请注意要将 sqljdbc4.jar 包放到 Tomcat 7 的 lib 目录下，以免因系统启动的先后原因找不到驱动程序。

如果是直接在 Eclipse 中对运行时环境中进行配置，则在“Project Explore”中选择目标服



务器中的配置文件 `server.xml`，比如本章的 Web 应用为 `ch06`，则在 `server.xml` 中找到以下的一行配置：

```
<Context docBase="ch06" path="/ch06" reloadable="true"
    source="org.eclipse.jst.jee.server:ch06"/>
```

修改成如下的配置：

```
<Context docBase="ch06" path="/ch06" reloadable="true"
    source="org.eclipse.jst.jee.server:ch06">
<Resource name="jdbc/sqlserver" auth="Container" type="javax.sql.DataSource"
    url="jdbc:sqlserver://localhost:1433;DatabaseName=testDatabase"
    username="sa" password="123" maxActive="5000" maxIdle="10" maxWait="-1" />
</Context>
```

`name` 属性配置的是连接池的 JNDI (Java Naming and Directory Interface) 名称，开发人员可以自行命名。`auth` 属性的值为“Container”表示生成连接池对象的工作由 Web 容器来完成。`type` 属性配置的是生成的数据源的类型，一般为“`javax.sql.DataSource`”。`driverClassName` 属性配置的数据库驱动程序的类名。`url` 属性用于配置连接数据的 URL。`username` 属性和 `password` 属性分别为登录数据的用户名和密码。`maxActive` 属性的值为连接数据库的最多活动连接数。`maxIdle` 属性的值为最多的空闲连接个数，值设为 -1 表示不作限制。`maxWait` 属性设置生成一个有效的数据库连接等待的时间，单位为秒 (s, second)，如果超过这个时间则表示超时了，将抛出超时异常，值设为 -1 表示不确定。

连接不同的数据库，读者可以根据本章 6.11 节中的相关内容来修改 `driverClassName`、`url`、`username`、`password` 参数的值即可。

`Resource` 的配置参数还有很多，表 6-4 列出了常用的一些配置参数。

表 6-4 Resource 的常用配置属性

属性名	默认值	说明
<code>username</code>	无	连接数据库的用户名
<code>password</code>	无	连接数据库的用户对应的密码
<code>url</code>	无	连接数据库的 URL
<code>driverClassName</code>	无	连接数据库的驱动程序类名
<code>connectionProperties</code>	无	连接数据库时需要设置的一些连接属性，采取“属性名=属性值;”的形式
<code>defaultAutoCommit</code>	<code>true</code>	生成的数据库连接是否自动提交
<code>defaultReadOnly</code>	驱动程序决定	生成的数据库连接是否只读，有些数据库可能不支持这个属性，如 Informix
<code>defaultTransactionIsolation</code>	驱动程序决定	默认的事务级别
<code>initialSize</code>	0	连接池创建时生成的数据库连接数
<code>maxActive</code>	8	连接池中最多的活动数据库连接数，0 或负数表示不限制
<code>maxIdle</code>	8	连接池中保留的最多的空闲数据库连接数，0 表示没有空闲的数据库连接
<code>minIdle</code>	0	连接池中保留的最少的空闲数据库连接数，负数表示不限制
<code>maxWait</code>	不确定	生成一个有效的数据库连接等待的时间，单位为秒 (s, second)，如果超过这个时间则表示超时了，将抛出超时异常，值设为 -1 表示不确定

## 6.12.2 Connection Pool 应用实例

### 【实例 6-12】利用连接池访问数据库

配置好数据库连接池后，就来编写应用程序。下面的程序是利用连接池取出数据库中用户表的所有用户数据。

ConnectionPool1.jsp

```
<%@ page contentType="text/html; charset=GB2312"%>
<%@ page import="java.sql.*"%>
<%@ page import="javax.naming.*"%>
<%
    try{
        Context initCtx = new InitialContext();
        Context ctx = (Context) initCtx.lookup("java:comp/env");
        //获取连接池对象
        Object obj = (Object) ctx.lookup("jdbc/sqlserver");
        //类型转换
        javax.sql.DataSource ds = (javax.sql.DataSource)obj;
        Connection conn = ds.getConnection();
        Statement stmt = conn.createStatement();
        String strSql = "SELECT * FROM userTable";
        ResultSet rs = stmt.executeQuery(strSql);
        out.print("<Table Border style='font-size: 10pt'>");
        out.print("<TR><td colspan=8 align=center>用户数据</td></tr>");
        out.print("<TR>");
        out.print("<Td width=60 >"+ "用户 ID 号</td>");
        out.print("<Td width=50 >"+ "用户名</td>");
        out.print("<Td width=100>"+ "用户真实姓名</td>");
        out.print("<Td width=40>"+ "年龄</td>");
        out.print("<Td width=40>"+ "性别</td>");
        out.print("<Td width=100>"+ "联系地址</td>");
        out.print("<Td width=100>"+ "联系电话</td>");
        out.print("<Td width=100>"+ "添加时间</td>");
        out.print("</TR>");
        while(rs.next())
        { out.print("<TR>");
            out.print("<TD >"+rs.getLong(1)+"</TD>");
            out.print("<TD >"+rs.getString(2)+"</TD>");
            out.print("<TD >"+rs.getString(4)+"</TD>");
            out.print("<TD >"+rs.getInt("user_age")+"</TD>");
            out.print("<TD >"+rs.getString("user_sex")+"</TD>");
            out.print("<TD >"+rs.getString("user_address")+"</TD>");
            out.print("<TD >"+rs.getString("user_telephone")+"</TD>");
            out.print("<TD >"+rs.getString("add_time")+"</TD>");
            out.print("</TR>");
        }
        out.print("</Table>");
    }catch(Exception ex){
        out.println(ex);
    }
%>
```



该程序首先通过 JNDI 找到 jdbc/sqlserver 对象, 使用 `Object obj = (Object) ctx.lookup("jdbc/sqlserver")`; 然后将得到的对象转换成 `DataSource` 类型, 进而得到连接, 得到连接后就可以进行相应的数据库操作。程序的运行结果如图 6-25 所示。



用户ID号	用户名	用户真实姓名	年龄	性别	联系地址	联系电话	添加时间
1	jiarong	邓佳容	30	女	湖南长沙高新区	8993421	2009-04-25 13:43:35.107
2	xingyu	刘新宇	34	男	湖南现代物流学院	3423343	2009-04-25 13:49:57.513
3	chengyun	程云	44	男	长沙中南大学	4433232	2009-04-25 13:50:30.42
4	xiaoming	刘小敏	34	女	首都师范大学	010-22332232	2009-04-25 14:21:34.077
6	liuyiping	刘一平	44	男	上海交通大学	029-3342233	2009-04-25 15:03:56.95
7	dengdt	邓地天	30	男	上海交通大学工学院	029-33422333	2009-04-25 15:04:32.56

图 6-25 利用数据库连接池访问数据库

## 6.13 得到元数据

有时在程序中需要得到元数据, 比如数据库中有什么表, 表又有什么字段, 字段又是什么数据类型, 这就要用到 `DatabaseMetaData` 和 `ResultSetMetaData` 了。

### 6.13.1 DatabaseMetaData

`DatabaseMetaData` 用于得到有关数据库的信息, 不过不同的关系 DBMS 常常支持不同的功能, 以不同方式实现这些功能, 并使用不同的数据类型。

有些 `DatabaseMetaData` 方法以 `ResultSet` 对象的形式返回信息列表。常规的 `ResultSet` 方法比如 `getString` 和 `getInt`, 可用于从这些 `ResultSet` 对象中检索数据。如果给定形式的元数据不可用, 则 `ResultSet` 获取方法抛出 `SQLException`。

有些 `DatabaseMetaData` 方法使用 `String` 模式的参数, 这些参数都有 `fooPattern` 这样的名称。在模式 `String` 中, “%” 表示匹配 0 个或多个字符的任何子字符串, “\_” 表示匹配任何一个字符。仅返回匹配搜索模式的元数据项。如果将搜索模式参数设置为 `null`, 则从搜索中删除参数标准。

`DatabaseMetaData` 常用的方法如下。

#### (1) `getURL()`

检索此数据库管理系统的 URL, 形式如下:

`String getURL() throws SQLException`

### (2) getUsername()

检索此数据库的已知的用户名称，形式如下：

```
String getUsername() throws SQLException
```

### (3) getTables()

检索可在给定类别中使用的表的描述。仅返回与类别、模式、表名称和类型标准匹配的表描述。它们根据 TABLE\_TYPE、TABLE\_SCHEMA 和 TABLE\_NAME 进行排序。方法形式如下：

```
ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern,
String[] types) throws SQLException
```

参数 catalog 表示类别名称，因为存储在数据库中，所以它必须匹配类别名称。该参数为 "" 则检索没有类别的描述，为 null 则表示该类别名称不应用于缩小搜索范围。

参数 schemaPattern 表示模式名称的模式，因为存储在数据库中，所以它必须匹配模式名称。该参数为 "" 则检索那些没有模式的描述，为 null 则表示该模式名称不应用于缩小搜索范围。

参数 tableNamePattern 表示表名称模式，因为存储在数据库中，所以它必须匹配表名称。

参数 types 表示要包括的表类型组成的列表，null 表示返回所有类型。

## 6.13.2 ResultSetMetaData

ResultSetMetaData 可用于获取关于 ResultSet 对象中列的类型和属性信息的对象。

### (1) getColumnCount()

用于返回此 ResultSet 对象中的列数，形式如下：

```
int getColumnCount() throws SQLException
```

### (2) getColumnName()

用于获取指定列的名称，形式如下：

```
String getColumnName(int column) throws SQLException
```

参数 column 表示列的编号，第 1 列为 1，第 2 列为 2，依此类推。

### (3) getColumnType()

用于检索指定列的数据库特定的类型名称，形式如下：

```
String getColumnType(int column) throws SQLException
```

参数 column 表示列的编号，第 1 列为 1，第 2 列为 2，依此类推。

## 6.13.3 得到表名和列名

下面来编写程序得到数据库中的表名和某个表的列名。

### 【实例 6-13】得到表名和列名

tableMeta.jsp

```
<%@ page contentType="text/html; charset=GB2312"%>
<%@ page import="java.sql.*"%>
<html>
```



```

<body>
<% Connection con;
Statement sql;
ResultSet rs;
try {
con=DriverManager.getConnection("jdbc:sqlserver:"+
    "///localhost:1433;DatabaseName=testDatabase","sa","123");
sql=con.createStatement();
//得到和显示表名
DatabaseMetaData dbMetaData = con.getMetaData();
String[] types = {"TABLE"};
ResultSet rsTable=dbMetaData.getTables(null, null, "%", types);
out.print("数据库中有以下表: <br>&nbsp;&nbsp;&nbsp;");
while(rsTable.next()){
    out.print(rsTable.getString(3)+"", " ");
}
out.print("<br>");
//显示 userTable 表的字段名和字段数据类型
rs=sql.executeQuery("SELECT * FROM userTable");
ResultSetMetaData rsMeta=rs.getMetaData();
out.print("userTable 表的字段名和数据类型如下: <br>");
for(int i=1;i<=rsMeta.getColumnCount();i++){
    out.print("&nbsp;&nbsp;&nbsp;"+rsMeta.getColumnName(i)+"字段的数据类型是: ");
    out.print(rsMeta.getColumnTypeName(i)+"<br>");
}
}catch(Exception ex){
    out.println(ex);
}
}%>
</body>
</html>

```

程序中用 Connection 对象的 getMetaData()方法得到了 DatabaseMetaData; 用 getTables()方法得到了所有表的结果记录集, 再用 while 循环显示出表的名称。通过 ResultSet 对象的 getMetaData()可得到 ResultSetMetaData; 用 ResultSetMetaData 的 getColumnCount()方法可得到表的列数, 用 getColumnName()可得到指定列的列名, 用 getColumnTypeName()可得到列的数据类型名称。

程序的运行结果如图 6-26 所示。

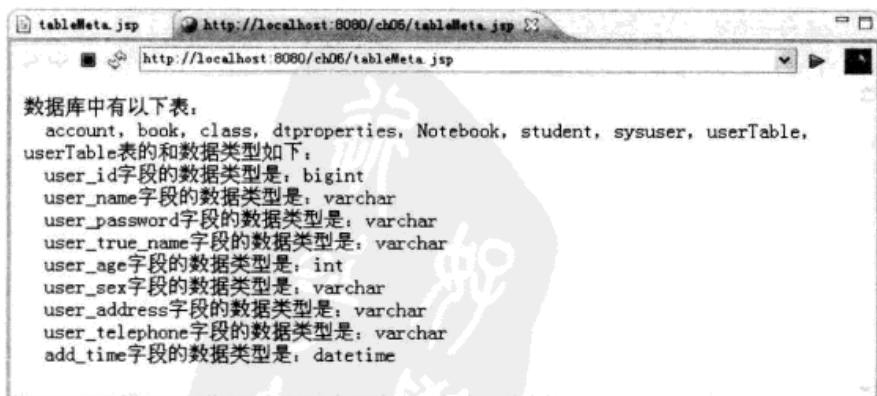


图 6-26 得到表名和字段名

## 6.14 小结

本章讲述了利用 JSP 使用数据库的方法，主要讲述了利用 JDBC-ODBC 方式和 JDBC 直接连接的方式。熟练掌握数据库的连接是开发基于 Web 的信息管理系统或网站的基础。

JSP 操作数据库是通过建立数据库连接对象后，利用连接对象向数据库提交 SQL 语句，因此先要有基本的 SQL 语句基础知识。在 JSP 中常用到的 SQL 语句有表操作语句，查询语句，插入、更新与删除语句以及存储过程。

Microsoft SQL Server 数据库连接的 JDBC 方式需要安装 JDBC 驱动。其他的数据库连接如果是 JDBC 方式，则相应地也需要安装驱动，实际上只需把相关的类包复制到 Web 服务器相应的目录下，在 Tomcat 中复制至 lib 子文件夹或 common\lib 文件夹均可。不同数据库的连接在程序设计时最大的区别就是驱动程序的加载和连接参数的设定。

用 DriverManager 的 getConnection() 方法可以创建一个数据库连接对象；一个数据库连接对象 Connection 表示与一个特定数据库的会话；Statement 类对象代表 SQL 语句，可用于将 SQL 语句发往数据库；PreparedStatement 类对象封装一条预编译的 SQL 语句；CallableStatement 类继承自 PreparedStatement 类，用于执行 SQL 存储过程；ResultSet 对象由生成它的语句自动关闭、再执行或从多个结果的序列中获取下一个结果。

调用存储过程来对数据库进行操作比直接调用 SQL 语句具有更高的性能和效率，存储过程存储在数据库服务器上，由数据库管理系统管理控制，并由服务器直接执行。

事务是一个独立的运行处理单元，具有原子性，Java 中用 Connection 对象的 commit() 方法提交事务，用 rollback() 方法回滚事务。

数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，可明显提高对数据库操作的性能。

通过运用 DatabaseMetaData 和 ResultSetMetaData 可以得到有关数据库的信息，比如表的名称、字段的名称和数据类型等。

## 6.15 练习

1. 在机器上安装 SQL Server 数据库系统，并安装 JDBC 驱动。
2. 在数据库中建立一个学生名册表，表名是 student，包括如表 6-5 所示的字段。

表 6-5 学生名册表字段说明

字段名	字段中文名	数据类型	是否主键	备 注
Student_id	学号	bigint	Yes	主键，自动生成，每增加一条记录自动增 1
Student_name	姓名	Varchar(20)	No	
Student_sex	性别	bit	No	
Student_class	班级	Varchar(4)	No	
Student_grade	年级	Varchar(4)	No	
Student_address	家庭住址	Varchar(60)	No	



建表成功后，至少输入 5 条记录。

3. 分别编写显示学生名册、增加学生记录、删除学生记录、修改学生信息的 JSP 程序，程序代码可参考本章中相关章节的内容，数据库中表的数据结构使用习题 2 中建立的表结构。

4. 在 Tomcat 7 服务器上配置一个数据库连接池，连接 SQL Server 数据库，并编写一个从数据库表中读取数据的程序（程序代码可参考实例 6-12）。



# 07

## JSP 中 JavaBean 的应用

---

JavaBean 是 Java 程序设计应用中的一种组件技术。本章从 JavaBean 的基本概念讲起，并说明如何编译 JavaBean，重点放在 Web 开发中 JavaBean 的开发、编译和部署上；后面给出了在 JSP 中如何使用 JavaBean 开发的应用，以及如何用 HTML 表单设置 Java 属性值等实例。

通过本章的学习，读者应当能够编写 JSP 开发中需要的 JavaBean，并能在 Tomcat 平台上部署。在本书的项目实战中还会大量地运用 JavaBean 技术。

### 7.1 什么是 JavaBean

Java 开发中的 JavaBean 就是一个类，用面向对象编程的思想封装了属性和方法，并用来完成某种特定功能的类。JSP 对于在 Web 应用中集成 JavaBean 组件提供了完善的支持。JavaBean 组件可以用来执行复杂的计算任务，或负责与数据库的交互以及数据提取等。

Java 中的 JavaBean 分为两种，一种是可视化 JavaBean，另一种是非可视化 JavaBean。可视化 JavaBean 是指带有界面的类，如：按钮、文本框等，在 C/S 模式的开发中多用此种 JavaBean，这种 Bean 类似于 VB、VC 开发中的控件；非可视化 JavaBean 是指在类的代码中没有界面的类，在 Web 开发中常用此种 JavaBean。因此本书主要讲述非可视化 JavaBean 的开发。使用 JavaBean 具有代码可重用的优点，可大大降低后续开发中程序员的劳动强度，能缩短开发时间，因为可以直接利用已有的经测试和可信任的组件，避免了重复开发。

JSP 开发中使用 JavaBean 可使 JSP 页面中的静态内容与动态内容较大程度地实现分离。如果用 JSP 来实现所有功能和业务处理，将使得 JSP 页面中的逻辑变得十分混乱，随着功能要求不断增强，程序代码越来越多，Java 代码和 HTML 代码错综复杂地交杂在一起，无论是程序的编写者还是代码的读者在查看起来时都会觉得头疼。为尽量减少 JSP 页面中 Java 代码的数据，用 JavaBean 来实现 Java 代码的功能是比较好的方式，可使页面中的程序逻辑变得清晰，编写程序和阅读修改程序都会变得容易一些。



## 7.2 编写 JavaBean

编写 JavaBean 实质上就是编写一个 Java 类，因此可以使用任何一个文件编辑器来编写，如记事本，只是使用专业的 Java 编程工具，可以减少代码的编写工作量，可提高编程的效率，如 JBuilder 中就提供了各种各样的向导来支持 Java 程序开发。

设计 JavaBean 类就是要设计这个 JavaBean 的属性和方法，类的方法命名遵循以下规则。

(1) 如果成员变量的名字是 `xxxx`，则相应地有两个用来得到成员变量值和设置变量值的方法，它们分别是 `getXxx()` 和 `setXxx()`。即如下的两种形式：

```
public dataType getXxx();
public void setXxx(dataType data);
```

其中，`dataType` 是成员变量的数据类型；参数 `data` 是赋予成员变量的值。



**注意** 这里 `getXxx()` 和 `setXxx()` 中变量名字的第一个字母为大写。

(2) 如果成员变量是 `boolean` 型数据，则有三种形式：

```
public boolean isXxx()
public boolean getXxx()
public void setXxx(boolean data)
```

前两种形式可用来得到成员变量的值；第三种形式用于设置成员变量的值，参数 `data` 是要设置的成员变量的值。

(3) 访问成员变量的方法都设为 `public`，即公有方法；如果有构造函数，则方法也为 `public` 型，并且无参数。

读者可能要问，遵循这些规则有什么用呢？这样可以方便 JSP 引擎知道 JavaBean 的属性和方法。

下面是一个 JavaBean 的源代码。

```
package maths;
public class Box{
    double length;//长
    double width;//宽
    double height;//高
    public Box(){//构造函数
        length=0;
        width=0;
        height=0;
    }
    public void setLength(double length){//设置长
        this.length=length;
    }
    public double getLength(){//得到长
        return length;
    }
    public void setWidth(double width){//设置宽
```

```

        this.width=width;
    }
    public double getWidth(){//得到宽
        return width;
    }
    public void setHeight(double height){//设置高
        this.height=height;
    }
    public double getHeight(){//得到高
        return height;
    }
    public double volumn(){//求容积
        double volumnValue;
        volumnValue=length*width*height;
        return volumnValue;
    }
    public double surfaceArea(){//求表面积
        double surfaceAreaValue;
        surfaceAreaValue=length*width*2+width*height*2+height*length*2;
        return surfaceAreaValue;
    }
}

```

编写完毕后，把这个文件保存为 **Box.java**。



**注意** 文件的名称要与类名相同，否则编译时会报错。

## 7.3 编译和部署 JavaBean

### 7.3.1 编译 JavaBean

在许多集成开发环境中，能够自动完成编译工作，如在 **Eclipse** 集成开发环境中编译 **JavaBean** 类的方法很简单，只需要保存源文件，即会自动编译（当然这需要在 **Eclipse** 环境中作出设置）。下面介绍手工编译的方法。

编译 **JavaBean** 实际上就是编译 **Java** 类。打开“命令与提示符工具”（在 **Windows** 的“开始”菜单中选择“运行...”，然后输入“**cmd**”，回车即可进入“命令与提示符工具”），切换到保存 **Box.java** 文件的目录下，输入如下命令。

```
javac Box.java
```

如果编译通过就会生成 **Box.class** 文件，这是 **JavaBean** 类的字节码文件。



**注意** 在编译前应配置好 **JAVA\_HOME** 系统变量，并加入 **path** 路径为 **JDK** 中 **bin** 的路径。如果没有加入路径可使用“**set path route**”命令，参数 **route** 是根据 **JDK** 中 **bin** 的路径来临时设置。

**javac** 是一个编译工具，在 **JDK** 安装目录的 **bin** 子目录下有 **javac.exe** 文件，它是这个编译器



的执行文件，也可以带一些参数来使用，形式如下。

```
javac [options] filename.java
```

其中，options 是编译时的选项，可以不设定；filename 是源程序文件的文件名；.java 是源程序文件的扩展名。javac 的选项有如下几种。

(1) -classpath path。设定编译时需要用到的 Java 类文件的路径，如果在系统变量 CLASSPATH 中已有则不必给出，参数 path 是类文件的路径。

(2) -d directory。设定编译生成的.class 文件输入到哪一个目录。通常情况下，javac 把生成的 .class 文件放在 .java 文件所在的目录中；如果使用 -d 参数，则可以指定 javac 将生成的 .class 文件在其他目录中；参数 directory 是要放入的目录。

(3) -g。此选项在代码产生器中打开调试表，以后可凭此调试产生字节代码。

(4) -nowarn。此选项禁止编译器产生警告。

(5) -o。告诉 javac 优化由内联的 static、final 以及 private 成员函数所产生的代码。

(6) -verbose。告知 Java 显示出有关被编译的源文件和任何被调用类库的信息。

最为常用的是第 1 项和第 2 项，设定要用到的类和 class 文件输出的路径。本例中就在与 Box.java 文件相同的目录中生成 class 文件并且无须使用其他的类文件，故没有设定选项。

如果使用 Eclipse 或 JBuilder 这样的可视化开发工具，无需用 javac 来编译，因为在 Eclipse 中保存 .java 文件即会自动编译为 .class 字节码（当然这需要在 Eclipse 环境中作出设置），并会按照类的层次结构生成对应的文件夹。

在前面的开发中曾用到过 jar 文件，那么 jar 文件到底是什么呢？一个 jar 文件是一个压缩包，可以包含一组类及其相关的资源，甚至是声音和图像文件，把这些文件组成统一的一个文件——jar 文件。有了 jar 文件，其中包含的类就不必一个一个地部署，可以作为一个 jar 文件一次部署，也可以在网上方便下载，因为下载一个文件比下载多个文件方便得多。

打包成 jar 文件之前先把要打成包的文件与资源放入同一个目录中，可以用子文件夹，再用 jar 工具来打包，其语法格式如下。

```
jar options [jarFileName.jar] files
```

其中，options 是打包时的选项，jarFileName 是打包后的 jar 文件名称，files 是要打包的文件。

options 选项较为常用的有：

c —— 创建一个新的存档文件。

f —— 文件列表中第一个文件的名称为要创建的或要访问的包文件的名称。

m —— 文件列表中的第一个文件是外部清单文件名。

t —— jar 文件的内容应制成表格。

u —— 更新已存在的 jar 文件。

v —— 当 JAR 工具执行时显示详细信息。

x —— 从 jar 文件中展开文件。

选项可以组合使用 files 为文件列表，也可以使用通配符，如果是所有文件用“\*”，如果是 JPEG 为扩展的则用 \*.jpg。如：创建一个名为 class 的 jar 文件，包括所有的 class 文件。

```
jar cf class.jar *.class
```

存在一个清单文件 Qd.mf, 可用:

```
jar cfm class.jar Qd.mf *.class
```

列出 class.jar 包中的文件:

```
jar tf class.jar
```

展开 class.jar 文件, 并释放在当前目录下:

```
jar xf class.jar
```

更新一个已存在的 class.jar 文件:

```
jar uf class.jar
```

## 7.3.2 部署 JavaBean

### 1. 部署 class

要部署 class, 则需要将这个 class 字节码文件复制到当前 Web 应用的“WEB-INF\classes”目录下, 如果 class 属于某个包, 则相应地应当位于“WEB-INF\classes”下的对应子目录中。

### 2. 部署 jar

如果要让 Web 服务器中所有的 JSP 页面都可以使用要部署的 jar 文件, 则可以把打包后的 jar 文件复制到 Tomcat 6 安装目录的 lib 子目录下。如果只对当前的应用有效, 则需在当前应用的 WEB-INF 子目录中建立一个新的子目录 lib, 并把 jar 文件复制过来, 即可在当前应用中使用 jar 文件中的类了。

### 3. 在 JSP 中应用 JavaBean

要在 JSP 页面中应用 JavaBean, 必须使用 JSP 指令标签 useBean。这个指令标签的语法如下:

```
<jsp:useBean id="给 JavaBean 实例取的名称" class="Bean 类名" scope="JavaBean 实例的有效范围"></jsp:useBean>
```

或:

```
<jsp:useBean id="给 JavaBean 实例取的名称" class="JavaBean 类名" scope="JavaBean 实例的有效范围"/>
```

其中, id 的设置可由用户任意给定; class 为 JavaBean 类名, 如果类之上还有包, 则此参数用形如“包名.类名”的形式; scope 有四种不同的取值范围, 分别解说如下。

scope 设为 page, 表示分配给每个客户的 JavaBean 不同, 有效范围仅对当前的 JSP 页面有效, 如果关闭此 JSP 页面, 相应的分配给此客户的 JavaBean 将被取消。

scope 设为 session 表示分配给每个客户的 JavaBean 不同, 但在同一个客户打开的多个 JSP 页面, 即一次会话其间, 是同一个 JavaBean。如果在同一客户的不同 JSP 页面中声明了相同 id 的 JavaBean, 且范围仍为 scope, 若更改此 JavaBean 的成员变量值, 其他页面中此 id 的 Bean 的成员变量值也会被改变。当客户打开服务器上的所有网页都被关闭时, 对应客户的这一次会话中的 JavaBean 也被取消。

scope 设为 request 表示分配给每个客户的 JavaBean 不同, 且有效范围在 request 期间,



即在请求与被请求页面之间共享 JavaBean。当对请求作出响应后, JavaBean 就会被取消。

scope 设为 application 表示在服务器的所有客户之间共享 JavaBean。当一个客户改变了成员变量的值时, 另一个客户的此 JavaBean 的同一个成员变量值也会被改变。当服务器关闭时 JavaBean 才会被取消。

在 JSP 页面中要能使用 JavaBean 还应事先在文件头部导入这个 JavaBean 对应的类, 如:

```
<%@page import="mathBox.Box"%>
```

其中, Box 是要导入的类名, 如果还有包, 可在类名前加包名, 形如“包名.类名”。如果在<jsp:useBean>指令的 class 属性设置中给出的是类的全路径名, 则可不用 import 语句导入。

### 【实例 7-1】JSP 中的 JavaBean 应用

假设前面编写的类 Box 位于 mathBox 包下, 经编译通过后已按本节中部署 class 文件或部署 jar 文件的方法部署好, 再来看在 JSP 页面中如何使用它。

#### javaBeanJSP1.jsp

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="mathBox.Box" %>
<HTML>
<BODY>
<jsp:useBean id="box" class=" mathBox.Box" scope="page">
</jsp:useBean>
<%
    box.setLength(10);
    box.setWidth(11);
    box.setHeight(12);
    out.println("在 JSP 中使用 JavaBean<br>");
    out.println("盒子的长度为: "+box.getLength()+"<br>");
    out.println("盒子的宽度为: "+box.getWidth()+"<br>");
    out.println("盒子的高度为: "+box.getHeight()+"<br>");
    out.println("盒子的容积为: "+box.volumn()+"<br>");
    out.println("盒子的表面积为: "+box.surfaceArea()+"<br>");
%>
</BODY>
</HTML>
```

该程序中首先用 import 语句把 mathBox.Box 类导入; 然后用 setXxxx()方法来设定类实例 box 的长宽高; 最后用 getXxxx()方法得到实例 box 的长宽高并输出, 用 volumn()方法和 surfaceArea()分别求出盒子的容积和表面积并输出。程序的运行结果如图 7-1 所示。

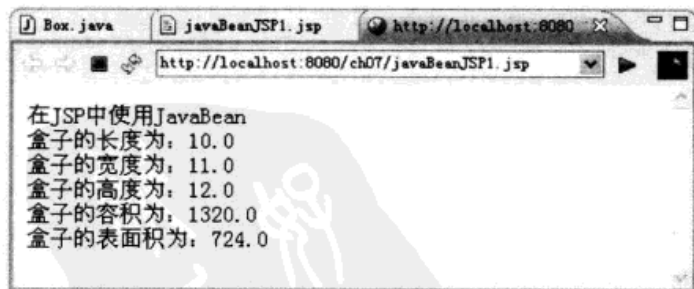


图 7-1 在 JSP 中使用 JavaBean

#### 4. 设置与得到 Bean 的属性值

使用 `setXxxx()` 方法和 `getXxxx()` 方法来设置和得到 `JavaBean` 的属性值，也可以用动作标签 `setProperty` 与 `getProperty`，分别用来修改与获取 `JavaBean` 的属性值。



**注意** 使用这两个标签之前应先导入 `JavaBean` 类和声明 `JavaBean` 的实例。

`getProperty` 标签获得 `JavaBean` 的属性值，并将这个属性值以字符串的形式显示出来。其语法格式如下：

```
<jsp:getProperty name="JavaBean 的名称" property="JavaBean 属性的名称" />或
<jsp:getProperty name="JavaBean 的名称" property="JavaBean 属性的名称">
</jsp:getProperty>
```

其中，`setProperty` 标签设置 `JavaBean` 的属性值，其具体的使用方法在第 4 章中已做过详细的介绍，在这里不再细说，读者可参见第 4 章中的相关内容。

应用 `setProperty` 标签设置 `JavaBean` 的属性值的方法有两种。第一种是直接设置为字符串或表达式，形如：

```
<jsp:setProperty name="JavaBean 的名称" property="属性名称" value="属性值" />
```

其中，属性值可以是字符串也可以是表达式，因为 Java 会把值转换为 `JavaBean` 中属性的数据类型，如果不能转换则会产生异常。

如实例 7-1 中的 JSP 文件程序代码可更改为如下的代码，两者的功能是等效的，这里使用的是 `setProperty` 标签和 `getProperty` 标签。

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="mathBox.Box" %>
<HTML>
<BODY>
<jsp:useBean id="box" class="mathBox.Box" scope="page">
</jsp:useBean>
<jsp:setProperty name="box" property="length" value="10"/>
<jsp:setProperty name="box" property="width" value="11"/>
<jsp:setProperty name="box" property="height" value="12"/>
在 JSP 中使用 JavaBean<br>
盒子的长度为: <jsp:getProperty name="box" property="length"/><br>
盒子的宽度为: <jsp:getProperty name="box" property="width"/><br>
盒子的高度为: <jsp:getProperty name="box" property="height"/><br>
<%
    out.println("盒子的容积为: "+box.volumn()+"<br>");
    out.println("盒子的表面积为: "+box.surfaceArea()+"<br>");
%>
</BODY>
</HTML>
```

第二种方法通过表单来设置 `JavaBean` 的属性值。这种方法要求 HTML 表单输入项的名字要与 `JavaBean` 属性的名字相同，这样服务器引擎会自动进行匹配把字符串转换为相应 `JavaBean` 属性的数据类型数据。使用如下的语法格式：

```
<jsp:getProperty name="JavaBean 的名称" property="*" />
```



## 【实例 7-2】用 HTML 表单设置 JavaBean 的属性值

javaBeanJSP2.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="mathBox.Box" %>
<jsp:useBean id="box" class="mathBox.Box" scope="page">
</jsp:useBean>
<jsp:setProperty name="box" property="**"/>
<html>
<head>
<title>用 HTML 表单设置 JavaBean 的属性</title>
</head>
<body>
<div align="center">
  <center>
    <table border="1" width="66%">
      <form name="form1" action="" method="post">
        <tr>
          <td width="44%">请输入盒子的长:</td>
          <td width="56%"> <input type="text" name="length" size="20"></td>
        </tr>
        <tr>
          <td width="44%">请输入盒子的宽:</td>
          <td width="56%"> <input type="text" name="width" size="20"></td>
        </tr>
        <tr>
          <td width="44%">请输入盒子的高:</td>
          <td width="56%"> <input type="text" name="height" size="20"></td>
        </tr>
        <tr>
          <td width="100%" colspan="2">
            <p align="center"><input type="submit" name="T1" size="20" value="提交">
              <input type="reset" name="T1" size="20" value="重置"> </td>
          </tr>
        </form>
        <tr>
          <td width="44%">你输入的盒子的长是:</td>
          <td width="56%"><%=box.getLength()%></td>
        </tr>
        <tr>
          <td width="44%">你输入的盒子的宽是:</td>
          <td width="56%"><%=box.getWidth()%></td>
        </tr>
        <tr>
          <td width="44%">你输入的盒子的高是:</td>
          <td width="56%"><%=box.getHeight()%></td>
        </tr>
      </table>
    </center>
  </div>
</body>
</html>

```

该程序中表单输入文本框的名称与 JavaBean 对象 box 的成员变量，即盒子的长、宽、高名称相同，故只需用 `<jsp:setProperty name="box" property="*" />` 语句就可设置三个成员变量的值。程序的运行结果如图 7-2 所示。

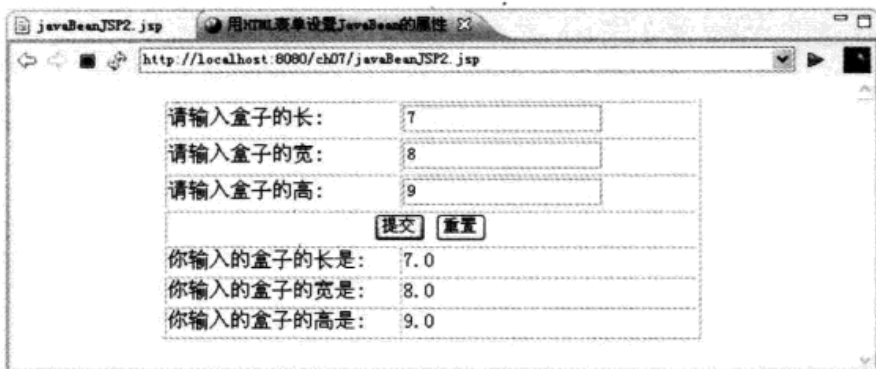


图 7-2 用 HTML 表单设置 JavaBean 的属性值

第三种方法用 `request` 的参数值来设置 JavaBean 的属性值。`request` 参数的名字和 JavaBean 属性的名字可以不同，服务器引擎会自动进行匹配把字符串转换为相应的 JavaBean 属性的数据类型数据，这样设置比第二种方法更为灵活。使用如下的语法格式：

```
<jsp:setProperty name="JavaBean 的名称" property="属性名称" param="request 参数的名字" />
```



**注意** 在 `<jsp:setProperty>` 标签中不能同时使用 `value` 和 `param`。

据此，把实例 7-2 中的 `<jsp:setProperty name="box" property="*" />` 语句更改为如下三个语句，可得到同样的效果：

```
<jsp:setProperty name="box" property="length" param="length" />
<jsp:setProperty name="box" property="width" param="width" />
<jsp:setProperty name="box" property="height" param="height" />
```

## 7.4 小结

本章介绍了 JSP 开发中 JavaBean 的开发与使用，掌握 JavaBean 的开发有利于提高 JSP 开发的效率和程序代码的可重复使用性，可使 JSP 页面中的静态内容与动态内容较大程度地实现分离。

JavaBean 是一个可重复使用的软件组件，可以用来执行复杂的计算任务，或负责与数据库的交互以及数据提取等。Java 中有两种 JavaBean，即可视化和非可视化 JavaBean，在 JSP 中一般使用后者。

编写 JavaBean 实质上就是编写一个 Java 类，编写属性和方法。编译 JavaBean 用 `javac` 命令，部署 JavaBean 只需将编译后的字节码文件复制到相应的“WEB-INF\classes”目录下即可在此应用中使用。



## 7.5 练习

1. 编写一个用户注册 JavaBean，在用户注册表单提交后，设置用户注册 JavaBean 的属性，并设方法实现在用户表中新增一条记录的功能。
2. 编写一个 JavaBean，功能是实现针对数据库某个表的增加、删除、更新及查询这些基本操作的封装，封装数据库中的哪个表请读者自定。

# 08

## Servlet 技术

---



本章将介绍 Servlet 的基本概念和工作原理；然后介绍用 Servlet 来获得表单的数据；再用 Servlet 来解决几个实际应用的问题：读写文件、访问数据库及在数据库中存取图片。

通过本章的学习，读者应当能够用 Servlet 来编写 HTML 和 Java 代码，并能够在 Web 平台上部署开发的 Servlet。

### 8.1 什么是 Servlet

Servlet 是用 Java 编写的运行在 Web 服务器中的程序，因此它可以调用服务器端的类，也可以被调用，它本身也就是一个类。



**注意** 要注意区分 Servlet、JavaScript 以及 Java Applet。JavaScript 是运行在客户端浏览器中的脚本程序；Java Applet 是运行在客户端的 Java 小程序；而 Servlet 是运行在服务器端的程序。

开发 Servlet 需要掌握较多的 Java 知识，虽然开发出来的 Java 程序功能强大，但开发时没有可视化界面及辅助工具辅助 Java 代码中 HTML 代码的编写，因此开发 Servlet 不及开发 JSP 程序效率高。

JSP 页面编写完毕后，在 Web 引擎中运行前也会被编译器先转换为 Servlet，再编译成字节码文件，因此 JSP 页面与 Servlet 是一一对应的。

但是，开发 Servlet 比开发 JSP 对程序员的要求更高，它需要程序员掌握更多的底层知识。一般在实际项目中常用 JSP 开发客户端界面，如：表单；用 Servlet 进行页面间的流程处理或编写一些操作性的非可视化程序代码。一种典型的开发模式 MVC 模式，就是用 Servlet 来控制程序的流程，这在后续章节中还会详细介绍。



## 8.2 Servlet 的工作原理

Servlet 由 Web 服务器引擎负责编译执行，当客户端浏览器访问 Servlet 时，服务器将启动一个线程来响应，而并非 CGI 技术的进程。因此相对 CGI 方式占用的系统资源（特别是内存）比较少，具有较高的运行效率。

当 Web 服务器中的 Servlet 被请求访问时，此 Servlet 被加载到 Java 虚拟机中，在 Servlet 中要接受 HTTP 请求并作相应处理。由于 Servlet 是在服务器端运行的，对客户完全透明，因此比 Java Applet 具有更好的安全性，当服务器有防火墙的保护时，Servlet 也受到防火墙的保护。

### 8.2.1 Servlet 的生命周期

一个 Servlet 的生命周期主要有以下三个过程。

(1) Servlet 的初始化。Servlet 实际是一个类，当第一次被客户端请求时，Web 服务器引擎首先要生成这个 Servlet 类的对象，并加载这个对象，通过这个对象的 `init()` 方法完成一些初始化的工作。

(2) 生成的 Servlet 类的对象调用 `service()` 方法来响应请求。

(3) Servlet 类的对象自第一次被生成后将常驻内存直至 Web 服务被关闭，当再次被请求时将直接从内存中取出对象来响应请求。当 Web 服务被关闭时，将调用 Servlet 类的对象的 `destroy()` 方法来消除此对象。

在上述的三个过程中，`init()` 方法只被调用一次，即第一次被请求时调用；`service()` 方法在每次 Servlet 被请求时都会被执行。

### 8.2.2 Servlet 相关的类及方法

要学会开发 Servlet 就需要熟悉与 Servlet 开发相关的 API。JSP 页面对应的 Servlet 继承了 `org.apache.jasper.runtime.HttpJspBase` 类，这个类是所有从 JSP 文件生成的 Servlet 类的父类。`org.apache.jasper.runtime.HttpJspBase` 类的继承情况如下所示。

```
java.lang.Object
├─ javax.servlet.GenericServlet
│   └─ javax.servlet.http.HttpServlet
│       └─ org.apache.jasper.runtime.HttpJspBase
```

可见 `org.apache.jasper.runtime.HttpJspBase` 类又继承自 `javax.servlet.http.HttpServlet` 类。`javax.servlet.http.HttpServlet` 类位于 `javax.servlet-api.jar` 包中，`javax.servlet-api.jar` 包位于 Tomcat 安装目录的 `lib` 子目录中，其中提供了 Servlet 开发的相关类、接口。

Tomcat 7 中使用的 Servlet 规范为 Servlet 3.0。Servlet 相关的 API 主要是两个包：`javax.servlet` 包和 `javax.servlet.http` 包。`javax.servlet` 包中提供了实现 Servlet 类的基类与接口，并与 Servlet 容器相容。`javax.servlet.http` 包中提供了基于 HTTP 协议实现 Servlet 类的基类与接口，并与 Servlet

容器相容。

要开发 Servlet 重点要掌握 `javax.servlet.http.HttpServlet` 类。`javax.servlet.http.HttpServlet` 类的声明情况如下：

```
public abstract class HttpServlet extends GenericServlet implements java.io.Serializable
```

可见在 `HttpServlet` 类中实现了 `Serializable` 接口，也就是说已经作了串行化处理，开发人员在继承 `HttpServlet` 类后不必再作串行化处理，简化了编程。此外，`HttpServlet` 类还继承了 `javax.servlet` 包中的 `GenericServlet` 类（通用的 Servlet 类）。

`HttpServlet` 类是一个抽象的类，用于创建一个基于 HTTP 协议的 Servlet，在继承它的类中必须实现以下方法中的至少一个。

- (1) `doGet()`：如果当前 Servlet 需要支持 HTTP GET 请求就需要覆盖并实现此方法。
- (2) `doPost()`：如果当前 Servlet 需要支持 HTTP POST 请求就需要覆盖并实现此方法。
- (3) `doPut()`：如果当前 Servlet 需要支持 HTTP PUT 请求就需要覆盖并实现此方法。
- (4) `doDelete()`：如果当前 Servlet 需要支持 HTTP DELETE 请求就需要覆盖并实现此方法。
- (5) `init()`和 `destroy()`：分别用于在 Servlet 生命周期的初始化时和销毁时用于管理相关的资源。
- (6) `getServletInfo()`：用于得到当前 Servlet 的一些相关信息。

此外还可以重载 `service()`方法，这个方法用于处理标准的 HTTP 请求，并将请求转发到各种请求类型对应的 `doXxxx()`方法（如果已定义），即 `doGet()`、`doPost()`、`doPut()`、`doDelete()`。其中最为常用的是 `doGet()`和 `doPost()`方法。也可以直接用 `service()`方法来响应请求，以适应各种不同的请求。

`HttpServlet` 类常用的方法有如下一些。

#### (1) `doGet()`

此方法通过 `service()`方法被调用，用于在 Servlet 中处理一个 GET 请求。在此方法中，开发人员常编写代码来读取请求（request）中的数据，完成响应（response）头中的设置，取得回应的输出流对象，再通过这个输出流对象来输出数据。应当在此方法中设置响应的 `content-Type` 以及 `encoding` 属性值。调用方法如下：

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, java.io.IOException
```

此外如果是使用 `PrintWriter` 对象来输出数据，应在使用之前就设置 `content-Type` 属性值，因为回应头必须在输出回应体的数据前作出设置。

参数 `req` 代表客户端的请求，`resp` 代表对客户端的响应。

#### (2) `doPost()`

此方法通过 `service()`方法被调用，用于在 Servlet 中处理一个 POST 请求。方法的其他说明同 `doGet()`方法。调用方法如下：

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, java.io.IOException
```

#### (3) `service()`

`service()`方法接收客户端的请求，并将请求转发给已定义的 `doXxxx()`方法。调用方法如下：



```
protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, java.io.IOException
```

#### (4) init()

此方法继承自 `GenericServlet` 类，用于对 `Servlet` 容器中的当前 `Servlet` 作初始化处理。调用方法如下：

```
public void init(ServletConfig config) throws ServletException
```

#### (5) destroy()

当前 `Servlet` 在生命周期结束时调用此方法。调用方法如下：

```
public void destroy()
```

### 8.2.3 部署 Servlet

要部署 `Servlet`，首先要编译 `Servlet`。编译 `Servlet` 的方法与编译 `JavaBean` 的方法一样，即执行 `javac` 命令，编译源代码文件 `.java` 为字节码文件 `.class`。如果使用 `Eclipse` 或 `JBuilder` 这样的可视化开发工具，无需用 `javac` 来编译，在 `Eclipse` 中保存 `.java` 文件即会自动编译为 `.class` 字节码（当然这需要在 `Eclipse` 环境中作出设置），并会按照类的层次结构生成对应的文件夹。

在 `Tomcat 7` 中编译 `Servlet` 要用到 `servlet-api.jar` 中相关的类，这个 `jar` 文件位于 `Tomcat 6` 的安装目录的“`lib`”子文件夹下，需要把它加入系统的类的路径中。

在桌面上的“我的电脑”图标上单击右键，选择“属性”快捷菜单，在弹出的“系统特性”对话框中选择“高级”选项卡，单击“环境变量”按钮，弹出“环境变量”对话框，在系统变量的 `CLASSPATH` 的值后加入如下值（`d:\tomcat7` 为 `Tomcat 7` 的安装目录）：

```
d:\tomcat7\lib\servlet-api.jar
```

类编译后，把字节码文件 `.class` 复制到当前应用的“`WEB-INF\classes`”目录下，即可使用了。这里注意 `Servlet` 类如果属于某个包，则在“`WEB-INF\classes`”相应的目录中。

复制了字节码文件 `.class` 后，还需在 `web.xml` 文件中进行配置。`web.xml` 位于当前应用的 `WEB-INF` 目录中，如果没有就自己建一个。在 `web.xml` 的 `<web-app>` 与 `</web-app>` 之间加入如下语句：

```
<servlet>
  <servlet-name>Servlet 名称</servlet-name>
  <servlet-class>Servlet 类</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Servlet 名称</servlet-name>
  <url-pattern>Servlet 名称访问路径</url-pattern>
</servlet-mapping>
```

其中，`<servlet-name>` 与 `</servlet-name>` 中配置的是 `Servlet` 的名称；`<servlet-class>` 与 `</servlet-name>` 中配置的是 `Servlet` 指向的类，形如 `packagename.classname`；`<url-pattern>` 与 `</url-pattern>` 中配置的是 `Web` 方式访问 `Servlet` 时相对当前应用的路径。

## 【实例 8-1】一个简单的 Servlet

FirstServlet.java

```
package userReg;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet{
    public void init(ServletConfig config) throws ServletException{
        super.init(config);
    }
    public void service(HttpServletRequest request,HttpServletResponse response)
    throws IOException{
        //设置 mime
        response.setContentType("text/html;charset=GB2312");
        PrintWriter out=response.getWriter();
        out.println("<HTML> <BODY>");
        out.println("这是一个简单的 Servlet。");
        out.println("客户端 IP 地址是: "+request.getRemoteAddr()+"<br>");
        out.println("</body> </html>");
    }
}
```

该程序中的 `init()` 方法直接使用父类的 `init()` 方法，用 `setContentType()` 设置成 MIME 类型，以支持中文显示；用 `HttpServletResponse` 类对象 `response` 的 `getWriter()` 得到输出流；用 `HttpServletRequest` 类对象 `request` 的 `getRemoteAddr()` 方法得到客户端的 IP 地址。

编译 Servlet 后，把 `FirstServlet.class` 文件复制到当前 Web 应用的“WEB-INF\classes”子目录中，并在 `web.xml` 的 `<web-app>` 与 `</web-app>` 之间加入如下语句：

```
<servlet>
    <servlet-name>firstServlet</servlet-name>
    <servlet-class>userReg.FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>firstServlet</servlet-name>
    <url-pattern>/firstServlet</url-pattern>
</servlet-mapping>
```

在浏览器中输入地址：

`http://localhost:8080/ch08/firstServlet`

即可运行 Servlet，其运行的结果如图 8-1 所示。

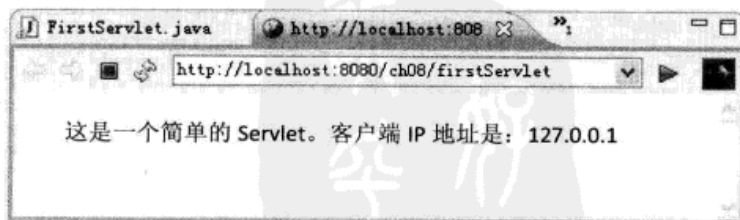


图 8-1 一个简单的 Servlet



## 8.3 用 Servlet 获取表单数据

### 【实例 8-2】用 Servlet 获取表单数据

我们延用第 5 章实例 5-2 中的表单来提交数据，再用 Servlet 来接收和显示数据。在提交表单的 JSP 页面中，仅作如下改动。

将 userRegist1.jsp 页面中的如下语句：

```
<form method="POST" action="acceptUserRegist1.jsp" name="form1"
    onsubmit="return on_submit()">
```

更改为：

```
<form method="POST" action="/ch08/acceptUserRegist" name="form1"
    onsubmit="return on_submit()">
```

更改的只是 action 参数，表示将表单的数据提交给 Servlet 来处理。

在 web.xml 文件的<web-app>与</web-app>中加入如下配置：

```
<servlet>
    <servlet-name>acceptUserRegist</servlet-name>
    <servlet-class>userReg.AcceptUserRegist</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>acceptUserRegist</servlet-name>
    <url-pattern>/acceptUserRegist</url-pattern>
</servlet-mapping>
```

AcceptUserRegist.java 源文件内容如下：

```
package userReg;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class AcceptUserRegist extends HttpServlet{
    public String codeToString(String str){//处理中文字符串的函数
        String s=str;
        try{
            byte tempB[]=s.getBytes("ISO-8859-1");
            s=new String(tempB);
            return s;
        }catch(Exception e){
            return s;
        }
    }
    public void init(ServletConfig config) throws ServletException{
        super.init(config);
    }
    public void doPost(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException{
        //设置 mime
        response.setContentType("text/html;charset=GB2312");
        PrintWriter out=response.getWriter();
        out.println("<HTML><head><title>接收新用户注册</title></head> <BODY>");
```

```

        out.println("这是新用户注册所提交的数据: <br>");
        out.println("用户名是: "+codeToString(request.getParameter
            ("username"))+"<br>");
        out.println("密码是: "+codeToString(request.getParameter
            ("userpassword"))+"<br>");
        out.println("性别是: "+codeToString(request.getParameter ("sex")) + "<br>");
        out.println("出生年月是: "+request.getParameter("year")+ request.getParameter
            ("month")+request.getParameter("day")+"<br>");
        out.println("电子邮箱是: "+request.getParameter("email")+"<br>");
        out.println("家庭住址是: "+codeToString
            (request.getParameter("address"))+ "<br>");
        out.println("</body> </html>");
    }
}

```

该程序中用 `doPost()` 方法来接收用 `post` 方法提交表单页面的数据，并将其显示在客户端，运行的结果如图 8-2 所示。这里的数据是在提交表单页面中输入的。

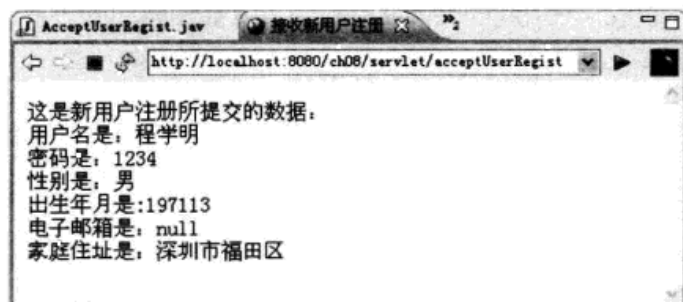


图 8-2 用 Servlet 接收表单数据

如果表单提交数据采用 `get` 方法，则相应的在 Servlet 中就应用 `doGet()` 方法来接收数据。

## 8.4 小结

本章系统地介绍了 Java Servlet 的基本原理与技术，以及相关的类与方法，并讲解了几个实例：获取表单数据、读写文件、访问数据库和在数据库中存取图片。

Servlet 是用 Java 编写运行在 Web 服务器中的程序，这点与 Java Applet 及 JavaScript 不同。Servlet 由 Web 服务器引擎负责编译执行。HTTP 协议下 Servlet 编程的 Servlet 类继承自 `javax.Servlet.http` 类，需重载 Web 请求的相关方法来对客户端进行响应。编写完 Servlet 后需用 `javac` 命令编译成字节码，并将字节码文件.class 部署后才能使用，而在 Eclipse 中开发则将这些工作交给 Eclipse 自动来完成就行了。

## 8.5 练习

1. 编写一个用户注册表单的 JSP 页面，提交给接收数据的一个 Servlet，并把用户数据加入数据库的用户表中。（程序代码可参考本章实例 8-1，用户表可使用本书中一直使用的用户表，读者也可自行在数据库中创建）。



# 09

## JSTL 应用开发

JSTL (JSP Standard Tag Library, JSP 标准标签库) 专为 Web 开发定制, 可用于页面中代码的基本输入/输出、流程控制、XML 文件处理及 SQL 处理等, 本章中将对这些内容进行介绍。

掌握 JSTL 是 JSP 程序员需要掌握的一种重要技术, 特别是在多层次式架构的 Web 信息系统中, JSTL 在表示层及在中间层对应用程序逻辑封装功能上表现优秀。通过本章的学习, 读者应当能够在 JSP 页面中熟练地运用和编写 JSTL 代码。

### 9.1 JSTL 技术概述

#### 9.1.1 JSTL 介绍

JSTL 由 Apache 的 jakarta 小组开发, 是开源的标准标签库, 目前正在不断完善中。使用 JSTL 将带来如下好处。

(1) 简化了 JSP 和 Web 程序的开发。原来许多需要大量的 Java 代码才能完成的功能, 现在用少量的 JSTL 标签即可完成, 而且 JSTL 标签具有良好的可读性, 易于理解, 不论是程序编写者还是其他阅读程序的人, 都容易理解 JSTL 标签的含义。

(2) 开发接口统一, 便于在各种服务器之间进行移植。

JSTL 标签库中的标签主要有如下 5 类 (库表示某一类标签)。

(1) 核心标签库。包括与表达式相关的标签, 输出 JSP 页面内容时的流程控制标签、迭代操作标签, 生成和操作 URL 的标签。

(2) XML 操作标签库。

(3) 格式化/国际化标签库。如: 数字和日期的格式化输出, 本地化资源在 JSP 页面中的国际化。

(4) 数据库操作标签库。

(5) 函数标签库。利用 EL (Expression Language, 表达式语言) 的 Function 实现, 主要用于处理字符串。

## 9.1.2 安装 JSTL

至本书成稿时，最新的版本是 JSTL1.1.2，可从如下的网址可下载得到：

[http://jakarta.apache.org/site/downloads/downloads\\_taglibs-standard.cgi](http://jakarta.apache.org/site/downloads/downloads_taglibs-standard.cgi)

从上面地址中的 `binaries` 子目录中可得到 Zip 文件。

解压缩此 Zip 包，在 `lib` 目录下可得到 `jstl.jar` 和 `standard.jar` 两个包，把这两个文件复制到当前 Web 应用的“WEB-INF\lib”目录中，JSTL 即在当前 Web 应用中可用；如果要在所有的 Web 应用中可用，可把这两个文件复制到 Tomcat 7 安装目录的 `lib` 目录下。

在解压得到的 `tld` 子目录中还有许多 `tld` 文件，这些是 JSTL 标签的描述文件，内容为 XML 格式。无须复制和配置这些 `tld` 文件，即可直接使用 JSTL，这是因为在 `standard.jar` 包的 `META-INF` 目录下已有这些 `tld` 文件。

## 9.1.3 标签书写语法约定

在本章中介绍使用 JSTL 的语法时，有如下约定：

- (1) [……]：用 “[” 和 “]” 括起来的内容表示可选项。
- (2) {选项 1|选项 2|……|选项 n}：表示  $n$  个选项中必选其一。
- (3) 语法描述时主要有两种表述方式：

第 1 种：<标签名 属性名="属性值" ... .../>

第 2 种：<标签名 属性名="属性值" ... ...>标签体</标签名>

第 2 种表述方式中的标签体也称为 `body`。

## 9.1.4 标签的分类

为方便区分各种 JSTL 标签，这里将 JSTL 的标签按功能类型作了分类规整，分为：核心标签库、XML 处理标签库、国际化处理标签库、SQL 标签库、函数标签库等。JSTL 标签的分类情况如表 9-1 所示。

表 9-1 JSTL 标签库

标签库名称	URI	使用时的前缀
核心标签库	<a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>	c
XML 处理标签库	<a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a>	x
国际化处理标签库	<a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a>	fmt
SQL 标签库	<a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>	sql
函数标签库	<a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a>	fn

当要使用不同的标签库中的 JSTL 标签时，根据表 9-1，需要在 JSP 页面的首部加入如下的语句：



```
<%@ taglib prefix="标签库使用时的前缀" uri="标签库的 URI" %>
```

如要使用核心标签库中的标签，则语句如下：

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

核心标签库、国际化处理标签库、SQL 标签库和函数标签库中的标签是常用的，在本章后续内容中将会作详细讲解，XML 处理标签库中的标签很少用到，因为处理 XML 文件大多在 Web 系统的后台 Java 代码中完成，很少需要直接在 JSP 页面中处理 XML 文件，所以本章暂不对 XML 处理标签库中的标签作详细说明。

## 9.2 核心标签

核心标签库中有最为常用的一些标签，包括：<c:out>、<c:set>、<c:remove>、<c:catch>、<c:if>、<c:choose>、<c:when>、<c:otherwise>、<c:forEach>、<c:forTokens>、<c:import>、<c:url>、<c:redirect>等。

### 9.2.1 表达式相关的核心标签

#### 1. <c:out>

<c:out>标签主要用来输出表达式的结果，比如：文本字符、数据库中查询出来的数据、session 变量等。使用<c:out>标签的语法格式如下：

第 1 种：<c:out value="表达式" [escapeXml="true|false"] [default="默认值"]/>

第 2 种：<c:out value="表达式" [escapeXml="true|false"]>默认值</c:out>

各属性的含义如表 9-2 所示。

表 9-2 <c:out>标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
value	Object	是	无	标签输出的值，可以是常量、变量、表达式或 EL 表达式
default	Object	否	无	当 value 属性值为 null 时，输出设置默认值
escapeXml	boolean	否	true	是否转换特殊字符，默认值为 true，如字符 "<" 会变为 "&lt;"

escapeXML 默认时值为 true，则转换表达式结果中的字符<、>、&、'、"为实体代码，设置为 false 则不转换。转换对应的情况如表 9-3 所示。

表 9-3 特殊字符转换对应情况

字 符	实体代码
<	&lt;
>	&gt;
'	&#039;
"	&#034;
&	&amp;

2. <c:set>

<c:set>标签用于设定指定范围内的一个变量或对象的值，使用的语法格式有如下 4 种：

第 1 种：使用 value 属性设置变量的值（无 body）：

```
<c:set value="要设置的值" var="变量名" [scope="{page|request|session|application}"]/>
```

第 2 种：使用 body 设置变量的值

```
<c:set var="变量名" [scope="{page|request|session|application}"]>
    要设置的值
</c:set>
```

第 3 种：使用 value 属性设置的对象的属性值（无 body）：

```
<c:set value="要设置的值" target="对象名称" property="对象的属性名称"/>
```

第 4 种：使用 body 设置对象的属性值：

```
<c:set target="对象名称" property="对象的属性名称">
    要设置的值
</c:set>
```

第 1 种和第 2 种语法用于设置指定范围内变量的值；第 3 种和第 4 种语法用于设置目标对象属性的值。如果目标对象是一个 java.util.Map 对象，则设置其中指定元素的值，如果这个元素不存在则新增一个；如果目标对象是一个 JavaBean 对象，如要设置的数据类型与属性的数据类型不匹配则会作自动转换，转换规则详见“10.1.4 自动类型转换”中的相关内容，转换失败则会抛出 JSPException 异常。

<c:set>标签的属性说明详见表 9-4。

表 9-4 <c:set>标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
value	Object	是	无	要被求值的表达式
var	String	是	无	要设置值的变量的名称
scope	String	否	page	变量的有效范围
target	Object	是	无	目标对象的名称。目标对象必须是一个 JavaBean 对象或 java.util.Map 对象
property	String	是	无	目标对象的属性名称，该属性必须带有 setXxxx()方法或目标对象是一个 java.util.Map 对象

在第 1 种语法中，如果 value 为 null，则将会删除变量的值。如果指定了 scope 属性，则删除变量值的方法相当于调用如下的 Java 语句：

```
pageContext.removeAttribute(变量名, scope);
```

如果没有指定 scope 属性，则删除变量值的方法相当于调用如下的 Java 语句：

```
pageContext.removeAttribute(变量名);
```

在使用第 3 种语法时，如果 value 为 null，也会作删除动作。如果目标对象是一个 java.util.Map 对象，则会根据 property 属性来匹配 java.util.Map 对象中对应名称的元素并作删除处理；如果



目标对象是一个 JavaBean 组件，则将此 JavaBean 组件的对应名称的属性值设为 null。

第 3 种语法和第 4 种语法在如下的情况下将会抛出异常。

(1) 目标对象为 null。

(2) 目标对象既不是一个 java.util.Map 对象也不是一个拥有对应的 setXxxx() 方法的 JavaBean 对象。

### 3. <c:remove>

<c:remove> 标签用于删除某范围内的一个变量，使用的语法格式如下：

```
<c:remove var="变量名" [scope="{page|request|session|application}"]/>
```

<c:remove> 标签的属性说明详见表 9-5。

表 9-5 <c:remove> 标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
var	String	是	无	被删除的变量的名字
scope	String	否	page	变量的有效范围

如果指定了 scope 属性，则删除变量值的方法相当于调用如下的 Java 语句：

```
pageContext.removeAttribute(变量名, scope);
```

如果没有指定 scope 属性，则删除变量值的方法相当于调用如下的 Java 语句：

```
pageContext.removeAttribute(变量名);
```

### 4. <c:catch>

<c:catch> 标签用于捕获嵌套在它里面的程序代码抛出的异常。使用 <c:catch> 标签的语法如下：

```
<c:catch [var="变量名"]>
    被嵌套的内容
</c:catch>
```

属性 var 是标识捕获异常的变量的名字。

## 【实例 9-1】表达式相关的核心标签综合实例

### useExpressJSTL.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>使用表达式相关标签</title>
</head>
<body>
    <c:set var="bookname" value="<<Java Web 极限开发>>" />
    变量 bookname 的值为: <c:out value="${bookname}" /><br>
    变量 bookname 的值为: <c:out value="${bookname}" escapeXml="false" /><br>
    <c:remove var="bookname" />
    变量 bookname 的值为: <c:out value="${bookname}" /><br>
    <c:catch var="errorStr">
```

```

        <%=3/0%>
        <c:out value="这是输出的内容"/>
    </c:catch>
    捕获到异常: <c:out value="${errorStr}"/>
</body>
</html>

```

程序运行的结果如图 9-1 所示。



图 9-1 表达式相关的核心标签综合实例

程序先用<c:set>标签将 bookname 变量的值设为“<<Java Web 极限开发>>”，再用<c:out>标签输出。默认情况下，escapeXml 的值为 true，故语句为：

```
<c:out value="${bookname}"/>
```

输出的 HTML 代码为：

```
&lt;&lt;Java Web 极限开发&gt;&gt;
```

故能够正常输出。如果将 escapeXml 的值设为 false，则相应输出的 HTML 代码为：

```
<<Java Web 极限开发>>
```

此时，浏览器会将这当作不可识别的标签，从而显示出“<>”。而后，再用<c:remove>删除了变量，故再次输出时没有内容。

<c:try>标签中的 Java 语句抛出了异常，标签将异常捕获放于变量 errorStr 中，再用<c:out>输出。由于产生了异常，<c:try>标签中嵌套的<c:out>标签语句根本没有执行。

## 9.2.2 流程控制核心标签

流程控制核心标签用于控制页面中的程序流程。

### 1. <c:if>

此标签的功能就像 Java 中的 if 判断语句，使用的语法格式如下：

第 1 种：

```
<c:if test="条件表达式" var="变量名" [scope="{page|request|session|application}"]/>
```

第 2 种：

```

<c:if test="条件表达式" [var="变量"] [scope="{page|request|session|application}"]>
    表达式成立后执行的代码
</c:if>

```



在第 2 种语法中，如果 scope 属性指定了，则 var 属性也必须给定。<c:if>标签的属性说明详见表 9-6。

表 9-6 <c:if>标签的属性


属 性	数据类型	是否必选项	默 认 值	属性值的说明
test	boolean	是	无	条件表达式，用于决定 body 中的代码是否执行
var	String	第 1 种语法中是，第 w2 种语法中不是	无	保存条件表达式结果的变量名称
scope	Strng	否	page	保存条件表达式结果的变量有效范围

2. <c:choose>、<c:when>及<c:otherwise>

<c:choose>标签用于进行排它性条件选择判断，这就像 Java 中的 switch 语句。使用<c:choose>标签的语法如下：

```
<c:choose>
  <c:when test="条件表达式 1">
    条件表达式 1 成立后执行的语句
  </c:when>
  [<c:when test="条件表达式 2">
    条件表达式 2 成立后执行的语句
  </c:when>]
  [...]
  [<c:otherwise>
    以上条件表达式都不成立后执行的语句
  </c:otherwise>]
</c:choose>
```

test 属性为条件表达式，用于决定<c:when>中的代码是否执行，结果为 true 则执行，为 false 则不执行。




**提示** <c:when>标签和<c:otherwise>标签必须嵌套在<c:choose>标签中使用，且在同一个<c:choose>标签中所有的<c:when>标签必须出现在<c:otherwise>标签之前。

9.2.3 迭代核心标签

JSTL 核心标签库中的迭代标签主要有<c:forEach>标签和<c:forTokens>标签。这两种标签用于完成一些循环处理功能，针对集合对象作迭代操作。

可以用<c:forEach>标签和<c:forTokens>标签作迭代处理的类范围广。可以是实现了 java.util.Collection 接口的类，包括 List、LinkedList、ArrayList、Vector、Stack、Set；可以是实现了 java.util.Map 接口的类，包括 HashMap、Hashtable、Properties、Provider、Attributes；可以是数组 arrays，数组中的数据如果是简单数据类型（如：int、float 等），<c:forEach>标签会自动将每一个元素当作标准的对应包装类（如 Integer、Float）对象来处理；可以是实现了 java.util.Iterator 和 java.util Enumeration 的类；可以是一个用逗号分隔的字符串（如：“Monday,Sunday,Tuesday”）。



提示 java.util.Map 中的元素实质上就是键值对，键（key）用于标识元素，值（value）是键对应的值。

1. <c:forEach>

<c:forEach>标签要么通过集合对象来决定执行循环体中嵌套的语句的次数，要么执行固定的次数。使用<c:forEach>标签的语法如下。

第 1 种：通过集合对象决定执行循环体的次数：


```
<c:forEach [var="变量名"] items="集合类对象" [varStatus="变量状态名"]
    [begin="起点"] [end="结束点"] [step="步长"]>
    循环体
</c:forEach>
```

第 2 种：执行固定次数的循环

```
<c:forEach [var="变量名"] [varStatus="变量状态名"]
    [begin="起点"] [end="结束点"] [step="步长"]>
    循环体
</c:forEach>
```

如果指定了 begin 值，则 begin 值必须大于或等于 0；如果指定的 end 值比 begin 值还要小，则循环体根本就不会执行；如果指定了 step 值，则 step 值必须大于或等于 1。在第 1 种语法中，如果 begin 比集合类对象的元素个数还要大，则循环体也不会被执行；如果集合类对象为 null，循环体也不会执行。

varStatus 中存放当前迭代的状态信息主要有 4 个属性：index，count，first，last。index 表示当前索引号，count 表示当前已迭代的次数，first 表示当前是否是第一次迭代，last 表示当前是否是最后一次迭代。



提示 items 中元素的下标从 0 开始。

<c:forEach>标签的属性说明详见表 9-7。

表 9-7 <c:forEach>标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
var	String	否	无	如果是第 1 种语法，则 var 属性中的变量代表了集合遍历时的当前元素，在<c:forEach>标签嵌套的语句中可通过这个名字得到当前元素；如果是第 2 种语法，则 var 属性中指定的变量值是迭代的次数
items	集合类	在第 1 种语法中是必选项	无	要进行迭代的集合类（collection）对象
varStatus	String	否	无	代表迭代的状态，可用来访问迭代本身的信息，是一个 javax.servlet.jsp.jstl.core.LoopTagStatus 对象
begin	int	否	0	如果是第 1 种语法，则从 items 的下标为 begin 的元素开始迭代；如果是第 2 种语法，则以 begin 作为迭代值的起点



续表

属 性	数据类型	是否必选项	默 认 值	属性值的说明
end	int	否	无	如果是第 1 种语法, 则从 items 的下标为 end 的元素处结束迭代; 如果是第 2 种语法, 则以 end 作为迭代值的终点
step	int	否	1	迭代的步长值

## 2. <c:forTokens>

<c:forTokens>标签专用于处理由某一符号分隔的字符串, 从其中拆分出子字符串。使用<c:forTokens>标签的语法如下:

```
<c:forTokens items="可拆分的字符串" delims="分隔符" [var="变量名"]
    [varStatus="当前迭代状态变量名"] [begin="起点"] [end="结束点"] [step="步长"]>
    循环体
</c:forTokens>
```

如果指定了 begin 值, 则 begin 值必须大于或等于 0; 如果指定的 end 值比 begin 还要小, 则循环体根本就不会执行; 如果指定了 step 值, 则 step 值必须大于或等于 1。如果 begin 比集合类对象的元素个数还要大, 则循环体也不会被执行; 如果集合类对象为 null, 循环体也不会执行。varStatus 的说明请参见<c:forEach>标签中的相关说明, 不再赘述。



**提示** 拆分后的数组元素的下标从 0 开始。

<c:forTokens>标签的属性说明详见表 9-8。

表 9-8 <c:forTokens>标签的属性

属 性	数据类型	是否必选项	默 认 值	属性的值的说明
var	String	否	无	迭代时用于存放当前拆分出的字符串的变量名称
items	String	是	无	可拆分的字符串
delims	String	是	无	拆分时使用的定界分隔符, 可以是一个字符, 也可以一个字符串
varStatus	String	否	无	当前迭代的状态信息
begin	int	否	0	用 delims 拆分 items 字符串后, 会得到一个字符串数组, begin 为这个数组的迭代开始下标
end	int	否	无	用 delims 拆分 items 字符串后, 会得到一个字符串数组, end 为这个数组的迭代结束下标
step	int	否	1	迭代的步长值

### 【实例 9-2】迭代核心标签综合实例

useForJSTL.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
```

```

<title>使用迭代标签</title>
</head>
<body>
<%//-----设置一个数组的初始值-----
String name[]=new String[4];
name[0]="黄少天";
name[1]="黄婧";
name[2]="邓佳容";
name[3]="孙学瑛";
request.setAttribute("name",name);
%>
-----用 forEach 输出数组元素值及迭代过程中的相关属性-----<br>
<c:forEach items="${name}" var="currentName" varStatus="currentVarStaus">
    当前元素值为:<c:out value="${currentName}"/>;
    当前元素索引号为:<c:out value="${currentVarStaus.index}"/>;
    当前共计迭代<c:out value="${currentVarStaus.count}"/>次;
    <c:if test="${currentVarStaus.first}">当前是第一次迭代操作</c:if>
    <c:if test="${currentVarStaus.last}">当前是最后一次迭代操作</c:if>
    <br>
</c:forEach><br>
-----用 forTokens 分拆字符串-----<br>
<%//-----设置一个字符串的初始值-----
String nameStr=new String("姓名=张明芳|年龄=18岁|性别=女");
request.setAttribute("nameStr",nameStr);
%>
字符串为:<c:out value="${nameStr}"/><br>
<!-------用 "|" 作分隔符----->
用|作分隔符<br>
<c:forTokens items="${nameStr}" var="currentName" varStatus="currentVarStaus"
    delims="|">
    当前元素值为:<c:out value="${currentName}"/>;
    当前元素索引号为:<c:out value="${currentVarStaus.index}"/>;
    当前共计迭代<c:out value="${currentVarStaus.count}"/>次;
    <c:if test="${currentVarStaus.first}">当前是第一次迭代操作</c:if>
    <c:if test="${currentVarStaus.last}">当前是最后一次迭代操作</c:if><br>
</c:forTokens>
</body>
</html>

```

程序的运行结果如图 9-2 所示。

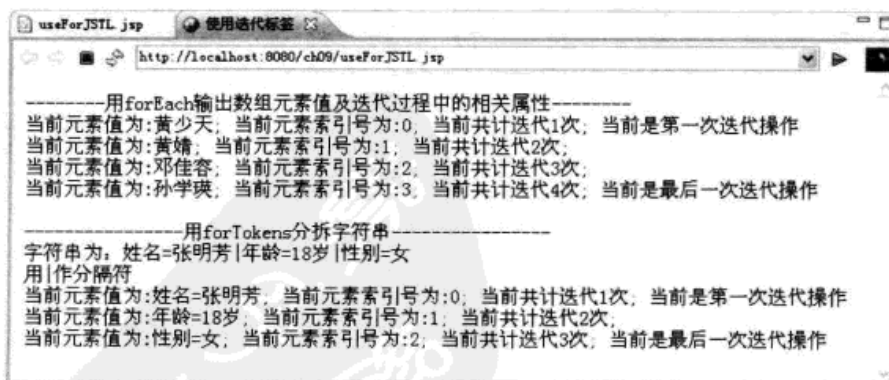


图 9-2 迭代核心标签综合实例



程序中首先是新建一个数组,并赋初始值;然后再把数组放入到 `request` 中去,在`<c:forEach>` 标签不断迭代的过程中,在浏览器中显示当前的数组元素值以及当前迭代的情况。

在用`<c:forTokens>`标签分拆字符串的代码中,初始化字符串值是用“|”将数据分隔开来的。`<c:forTokens>`标签仅用“|”作分隔符,故 `delims` 的属性值为“|”。

## 9.2.4 URL 相关的核心标签

JSP 页面中经常需要做 URL 资源的链接、导入、重定向等处理,JSTL 核心标签中有一些 URL 相关的标签:`<c:import>`、`<c:url>`、`<c:redirect>`、`<c:param>`。

### 1. `<c:import>`

`<c:import>`标签用于将其他文件包含到当前 JSP 文件中,其他文件可以是动态文件也可以是静态文件。使用`<c:import>`标签的语法如下:

```
<c:import url="URL" [var="变量名"] [scope="{page|request|session|application}"]
    [charEncoding="字符编码"]>
    [<c:param>子标签内容]
</c:import>
```

`<c:import>`标签的属性说明详见表 9-9。

表 9-9 `<c:import>`标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
url	String	是	无	要导入的 URL 资源
var	String	否	无	存放外部资源内容的变量
scope	String	否	page	var 属性中指定的变量的有效范围
charEncoding	String	否	无	导入的资源内容的字符编码

导入文件时,把文件的内容以 `String` 类型存入指定的变量(`var` 属性指定的变量)。URL 可以是网址,如“`http://www.56edu.com`”,也可以是 FTP 服务器上的文件地址,如“`ftp://ftp.56edu.com/top.txt`”,还可以是当前 Web 应用或本服务器上的其他应用中的文件。`urlString` 如果以“/”开头,表示相对 Web 服务器的根目录的路径。假设当前在 Web 应用 `jstl` 的主目录下,要导入 `allinclude` 子目录下 `top.txt` 文件时,URL 可以是“`/jstl/allinclude/top.txt`”或“`allinclude/top.txt`”。

`<c:param>`标签语句用于向导入的页面中传入参数,其用法如下:

```
<c:param name="参数名" value="参数值"/>
```

其中,属性 `name` 设置的是传入的参数名称,属性 `value` 设置的是属性 `name` 指定的参数对应的值。

如果属性 `url` 为 `null`、`empty` 或是无效的,将会抛出一个 `JspException` 异常。

### 2. `<c:url>`

`<c:url>`标签用于生成一个 URL。语法格式如下:

第 1 种：没有 body 时

```
<c:url value="生成的 URL" [context="外部 context 资源的名字"]
    [var="变量名"] [scope="{page|request|session|application}"]/>
```

第 2 种：带有 body，并给定一些特定的参数

```
<c:url value="value" [context="外部 context 资源的名字"]
    [var="生成的 URL "] [scope="{page|request|session|application}"]>
    [<c:param> 子标签]
</c:url>
```



**提示** 如果指定了 context，则 context 和 url 属性中的字符串都应以 “/” 开头。

第 1 种与第 2 种语法的区别就在于：第 2 种语法中生成的 URL 可通过<c:param>标签语句增加 URL 中的参数。context 属性用于 URL 为其他 Web 应用中的网页的情况，它指出应用名。如：当前 Web 应用为 jstl，要生成指向 Web 应用 xml 中 parseXML.jsp 文件的 URL 时，其语句为：

```
<c:url value="/parseXML.jsp" context="/xml"/>
```

<c:url>标签的属性说明详见表 9-10。

表 9-10 <c:url>标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
value	String	是	无	要被处理的 URL
context	String	否	无	外部 context 名称
var	String	否	无	存放生成的 URL 字符串的变量
scope	String	否	page	var 属性中指定的变量的有效范围

### 3. <c:redirect>

<c:redirect>标签用于把客户端发来的请求重定向到另一个页面，语法格式如下。

第 1 种：没有 body 时：

```
<c:redirect url="URL" [context="外部 context 资源的名字"]/>
```

第 2 种：有 body 并带一些特定的参数

```
<c:redirect url="value" [context="外部 context 资源的名字"]/>
    [<c:param>子标签]
</c:redirect>
```

<c:redirect>标签的属性说明详见表 9-11。

表 9-11 <c:redirect>标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
url	String	是	无	要被导向的资源的 URL
context	String	否	无	外部 context 名称

第 1 种与第 2 种语法的区别在于：第 2 种语法可以向目标 URL 通过<c:param>标签带入参数。URL 是重定向的目标网页，context 用于当要重定向的目标网页为其他 Web 应用的网页时



指出应用名。如：当前 Web 应用为 jstl，要重定向到 Web 应用 xml 中的 parseXML.jsp 文件时，其语句为：

```
<c:redirect url="/parseXML.jsp" context="/xml"/>
```

### 【实例 9-3】使用 URL 标签

useURLJSTL.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>使用 URL 标签</title>
</head>
<body>
    -----用 url 标签生成一个 URL-----<br>
    <c:url value="test.jsp" scope="page" var="testURL">
        <c:param name="collegeName" value="湖南现代物流职业技术学院"/>
    </c:url>
    生成的 URL 为:<c:out value="${testURL}"/>
</body>
</html>
```

程序的运行结果如图 9-3 所示。

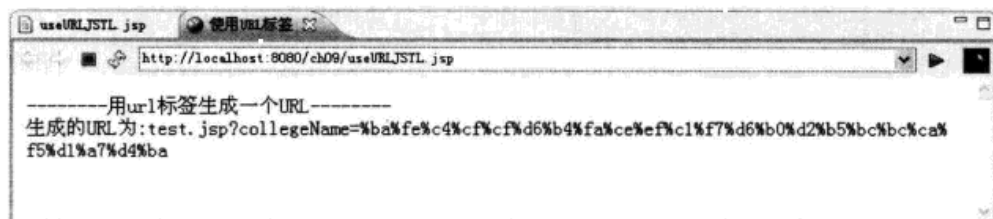


图 9-3 使用 URL 标签

在生成的 URL 中，显示 collegeName 为乱码“= %ba%fe%c4%cf%cf%d6%b4%fa%ce%ef%c1%f7%d6%b0%d2%b5%bc%bc%ca%f5%d1%a7%d4%ba”，这是因为 collegeName 的值为中文字符，需要在接收参数的页面中做字符集处理，可转换为 GBK 字符集或 GB2312 字符集，这样就不会出现乱码了。

## 9.3 国际化处理标签

JSTL 的国际化格式标签能让数据适应不同的国家、语言、文化要求，分为国际化类、消息类、数字日期格式化类 3 种 24 个，如下所示。

- (1) 国际化类：<fmt:setLocale>、<fmt:requestEncoding>。
- (2) 消息类：<fmt:bundle>、<fmt:message>、<fmt:setBundle>、<fmt:param>。
- (3) 数字日期格式化类：<fmt:formatNumber>、<fmt:formatDate>、<fmt:parseDate>、<fmt:parseNumber>、<fmt:setTimeZone>、<fmt:timeZone>。

要使用国际化格式标签库必须在 JSP 文件首部加入如下语句：

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

### 9.3.1 国际化类处理标签

#### 1. <fmt:setLocale>

<fmt:setLocale> 标签用于设置用户的语言与国家（或地区），仅对 <fmt> 标签生效。

<fmt:setLocale> 标签的使用语法如下。

```
<fmt:setLocale value="语言与国家（或地区）" [variant="浏览器类型"]  
[scope="{page|request|session|application}"]/>
```

<fmt:setLocale> 标签的属性说明详见表 9-12。

表 9-12 <fmt:setLocale> 标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
value	String 或 java.util.Locale	是	无	用户的语言与国家（或地区）设置
variant	String	否	无	浏览器类型设置
scope	String	否	page	用户的语言与国家（或地区）设置的有效范围

其中，属性 value 设置的是语言与国家（或地区）代码，可以是语言代码，如：zh、ch；也可在其后再加上两位国家（或地区）代码，中间用“-”或“\_”连接起来，如：zh\_TW，即为中文（中国台湾地区）。属性 variant 用于设置浏览器的类型，如：WIN 代表 Windows、Mac 代表 Macintosh。属性 scope 设置国家（或地区）的有效范围，默认为 page。如果属性 value 中的值为 null 或 empty，将使用 Web 容器默认的语言与国家（或地区）代码设置。

常用的语言与国家（或地区）代码如表 9-13 所示。

表 9-13 常用的语言与国家（或地区）代码

语言及国家（或地区）代码	语言及国家（或地区）名称
en	英文
zh_CN	中文
fr	法语
de	德语
ja	日语

#### 2. <fmt:requestEncoding>

<fmt:requestEncoding> 标签用于设置请求中数据的字符集，使用 <fmt:requestEncoding> 标签的语法如下：

```
<fmt:requestEncoding value="字符集名称"/>
```

其中，参数 charsetName 是要设置的字符集名称。这在中文处理时特别有用，程序中将不必再为 request 请求发过来的每个参数作字符编码转换，只需如下的语句：

```
<fmt:requestEncoding value="gb2312"/>
```



### 9.3.2 消息类处理标签

#### 1. <fmt:bundle>

<fmt:bundle>标签用于绑定消息资源。<fmt:bundle>标签的使用语法如下：

```
<fmt:bundle basename="资源文件名" [prefix="消息前缀"]>
    处理消息的语句或其他程序语句
</fmt:bundle>
```

这个标签根据属性 `basename` 指定的资源文件名去查找 `.properties` 文件，并作绑定处理。Web 容器将处理 <fmt:bundle> 标签的 `body` 中的语句并作输出，<fmt:bundle> 标签本身并不处理 `body` 中的语句。`prefix` 指出在 `body` 中可能要作处理的消息的前缀。如果属性 `basename` 的值为 `null` 或 `empty`，或要绑定的资源文件找不到，将装载一个为 `null` 的资源绑定。

在 `.properties` 文件中如果有中文，读取时将会显示乱码，要解决这个问题，就必须对 `.properties` 文件作编码转换。在 JDK 中提供了转换的工具——`native2ascii.exe`，此工具位于 JDK 安装目录的 `bin` 目录中。查看系统变量 `path` 中是否设置了 JDK 安装目录的 `bin` 目录，如果没有则将其添加上。或在使用 `native2ascii.exe` 工具前执行如下命令：

```
set path=%path%;%JAVA_HOME%/bin;
```



**提示** 执行这个命令要与使用 `native2ascii.exe` 工具的命令在同一个命令会话（即打开的同一个 `command` 窗口）中。

用 `native2ascii.exe` 工具执行如下命令：

```
native2ascii -encoding gb2312 源.properties 文件名 目标.properties 文件名
```

如要把当前目录下的 `dbconn.properties` 文件进行转码，转码后的文件名为 `dbconn_zh.properties`，其命令为：

```
native2ascii -encoding gb2312 dbconn.properties dbconn_zh.properties
```

#### 2. <fmt:message>

<fmt:message>标签用于从指定的资源文件中把指定的键值取出来。<fmt:message>标签的使用语法如下。

第1种：不带 `body` 时

```
<fmt:message key="消息的键名" [bundle="绑定的资源文件名"]
    [var="变量名"] [scope="{page|request|session|application}"]/>
```

第2种：带特定的消息参数 `body` 时

```
<fmt:message key="消息的键名" [bundle="绑定的资源文件名"]
    [var="变量名"] [scope="{page|request|session|application}"]>
    [<fmt:param>标签语句]
</fmt:message>
```

<fmt:message>标签的属性说明详见表 9-14。

表 9-14 &lt;fmt:message&gt;标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
key	String	是	无	要在资源文件中查找的消息键名称
bundle	LocalizationContext	否	无	要查找的键所在的要被绑定的资源或变量
var	String	否	无	存放查找出的消息的变量
scope	String	否	page	属性 var 指定的变量的有效范围

如果属性 `scope` 被指定了则属性 `var` 必须被指定。如果设置了属性 `var` 的值，则使用此标签时不会在 JSP 页面中显示键值，需要用 `<c:out>` 标签输出。

如果 `key` 的值为 `null` 或 `empty`，则会当作消息在资源文件中找不到的情况来处理，会在浏览器中显示“?????”；如果 `key` 中指定的键值在资源文件中找不到，会显示“???key 名称???”。

`<fmt:param>` 标签需要与 `<fmt:message>` 标签配合使用，用于设定 `<fmt:message>` 标签指向键的动态值。



**提示** 资源文件键值中动态参数的编号从 0 开始。

### 3. <fmt:setBundle>

`<fmt:setBundle>` 标签用于创建一个资源绑定，并把这种绑定存放到一个变量中。`<fmt:setBundle>` 标签的使用语法如下。

```
<fmt:setBundle basename="资源文件名" [var="变量名称"]
    [scope="{page|request|session|application}"] />
```

`<fmt:setBundle>` 标签的属性说明详见表 9-15。

表 9-15 &lt;fmt:setBundle&gt;标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
basename	String	是	无	绑定的资源文件名
var	String	否	无	代表绑定的资源的变量名称
scope	String	否	page	属性 var 中指定的变量的有效范围

如果属性 `basename` 的值为 `null` 或 `empty`，或要绑定的资源文件找不到，将装载一个为 `null` 的资源绑定。



**提示** `<fmt:setBundle>` 标签与 `<fmt:bundle>` 标签是有所区别的，前者把资源绑定的关系存放在变量中，这样就可以在标签之外的其他处使用绑定的资源；后者使用绑定的资源必须是在标签体内。

## 9.3.3 数字日期格式化类处理标签

### 1. <fmt:formatNumber>

`<fmt:formatNumber>` 标签用于对数字、货币、百分比数据作格式化处理。使用



<fmt:formatNumber>标签的语法如下。

第 1 种：不带 body

```
<fmt:formatNumber value="数值" [type="{number|currency|percent}"]
    [pattern="格式定制模式"] [currencyCode="货币代码"]
    [currencySymbol="货币符号"] [groupingUsed="{true|false}"]
    [maxIntegerDigits="最多的整数位数"] [minIntegerDigits="最少整数位数"]
    [maxFractionDigits="最多的小数位数"] [minFractionDigits="最少的小数位数"]
    [var="变量名"] [scope="{page|request|session|application}"]/>
```

第 2 种：带有 body

```
<fmt:formatNumber [type="{number|currency|percent}"] [pattern="格式定制模式"]
    [currencyCode="货币代码"] [currencySymbol="货币符号"]
    [groupingUsed="{true|false}"] [maxIntegerDigits="最多的整数位数"]
    [minIntegerDigits="最少整数位数"] [maxFractionDigits="最多的小数位数"]
    [minFractionDigits="最少的小数位数"] [var="变量名"]
    [scope="{page|request|session|application}"]>
    要被格式化处理数字
</fmt:formatNumber>
```

<fmt:formatNumber>标签的属性说明详见表 9-16。

表 9-16 <fmt:formatNumber>标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
value	String 或数字	是	无	要被格式化的数值
type	String	否	number	指定被格式化的数值的数据类型，只能是 number、currency 或 percent 中的一种
pattern	String	否	无	定制的格式模式
currencyCode	String	否	无	ISO 4217 标准中的货币代码，仅当格式化货币数据类型时有效
currencySymbol	String	否	无	货币符号，如 ¥；仅当格式化货币数据类型时有效
groupingUsed	boolean	否	true	是否输出分隔符，如：1,234,567
maxIntegerDigits	int	否	无	整数部分最多的整数位数
minIntegerDigits	int	否	无	整数部分最少整数位数
maxFractionDigits	int	否	无	小数部分最多的小数位数
minIntegerDigits	int	否	无	小数部分最少的小数位数
var	String	否	无	存储格式化处理后输出的结果字符串的变量
scope	Strng	否	page	属性 var 中指定的变量的有效范围

如果属性 scope 被指定了，则属性 var 也必须被指定。属性 currencyCode 中设置的值必须是 ISO 4217 标准中规定的有效代码。如果属性 value 中的值为 null 或 empty，则不会作输出处理，即便是指定了 var 属性也会从 scope 属性指定的范围中把这个变量删除。如果格式化处理失败，则会将要格式化的数值转化为字符串输出。指定了属性 var 则不会作输出处理，只是把格式化的结果存入属性 var 指定的变量中。

如果处理的数据类型是 percent，即百分比，则数值会被乘以 100，再根据本地化设置来作

输出处理，数值为“.24”表示“24%”，数值为“24”表示“2400%”。如下面的语句：

```
<fmt:formatNumber type="percent" value="24"/>
```

在美国区域设置下，输出为“2,400%”，但在法国区域设置下，输出为“2400%”。

货币数据有两个重要的特性：

- (1) 有货币符号，如美元为“\$”，人民币为“¥”，法郎为“F”。
- (2) 小数点后的位数有特定的标准，如人民币和美元是小数点后 2 位，但意大利里拉是不能带小数的。

如下面的语句：

```
<fmt:formatNumber type="currency" value="78.74901"/>
```

对于人民币输出为“¥78.75”，对于意大利里拉输出为“L.79”。

一般情况下，使用系统默认的货币代码即可，如果需要设置特定的货币代码，就要设置属性 `currencyCode` 的值，如：“USD”表示美元。常用的货币代码见表 9-17 所示。

表 9-17 常用的货币代码

货币代码	货 币
CNY	人民币元
EUR	欧元
GBP	英镑
JPY	日圆
USD	美元

需要更多的货币代码，可参见如下的网址：

<http://www.xe.com/iso4217.php>

属性 `groupingUsed` 指出格式化数据时，是否加入分隔符，默认情况下是加入的。如下的语句：

```
<fmt:formatNumber value="500000.01" groupingUsed="true" />
```

在英国区域设置下，输出为“500,000.01”。但如下的语句：

```
<fmt:formatNumber value="500000.01" groupingUsed="false" />
```

在英国区域设置下，输出为“500000.01”。

`maxIntegerDigits`、`minIntegerDigits`、`maxFractionDigits`、`minFractionDigits` 这 4 个属性用于设置数字位数。

如果整数部分位数少于 `minIntegerDigits`，将在左边补 0；如果多于 `maxIntegerDigits`，将会截去前面多的位数。如果小数部分位数小于 `minFractionDigits`，将在右边补 0；如果多于 `maxFractionDigits`，则会作四舍五入处理。

如数字“99.2”，根据上述 4 个属性设置格式化之后的结果见表 9-18 所示。

表 9-18 数字位数控制情况示例

<code>minIntegerDigits</code>	<code>maxIntegerDigits</code>	<code>minFractionDigits</code>	<code>maxFractionDigits</code>	结 果
		2		99.20
		3		99.200



续表

minIntegerDigits	maxIntegerDigits	minFractionDigits	maxFractionDigits	结 果
		4		99.2000
			0	99
			1	99.2
			2	99.2
			3	99.2
4		4		0099.2000
	1		1	9.2
2	4	2	4	99.20

属性 `pattern` 可用于设置数值的显示风格, 在针对大的数字需要作科学计数法处理时特别有用。如 “###.###E0”。有如下的语句:

```
<fmt:formatNumber value="203787490020343266877275964040"
    pattern="###.###E0" />
```

输出为: “203.787E27”。

## 2. `fmt:formatDate`

`<fmt:formatDate>` 标签用于格式化处理日期或时间数据。使用 `<fmt:formatDate>` 标签的语法如下:

```
<fmt:formatDate value="日期或时间" [type="{time|date|both}"]
    [dateStyle="{default|short|medium|long|full}"]
    [timeStyle="{default|short|medium|long|full}"]
    [pattern="定制的格式模式"] [timeZone="时区设置"]
    [var="变量名"] [scope="{page|request|session|application}"] />
```

`<fmt:formatDate>` 标签的属性说明详见表 9-19。

表 9-19 `<fmt:formatDate>` 标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
value	java.util.Date	是	无	要被格式化处理的日期或时间数据
type	String	否	无	数据类型, 为 time (时间)、date (日期) 或 both (日期与时间的组合)
dateStyle	String	否	default	格式化日期的风格
timeStyle	String	否	default	格式化时间的风格
pattern	String	否	无	定制数据的格式模式
timeZone	String 或 java.util.TimeZone	否	无	数据格式的时区设置
var	String	否	无	存放格式化后的结果的字符串
scope	String	否	page	属性 var 中指定的变量的有效范围

如果属性 `scope` 被指定了, 则属性 `var` 也必须被指定。如果属性 `value` 中的值为 null 或 empty, 则不会作输出处理, 即便是指定了 `var` 属性也会从 `scope` 属性指定的范围中把这个变量删除。

对于不同的时区, 采用不同的 `timeStyle` 和 `dateStyle` 属性设置, 格式化后结果可能并不相

同。如时间“2007 年 6 月 20 日 早上 7: 51: 30”，时区为英国，情形如表 9-20 所示。

表 9-20 不同的属性 timeStyle 和 dateStyle 设置对格式化的影响

属 性 值	日期格式化后的结果	时间格式化后的结果
default	Jun 20, 2007	7:51:30 AM
short	6/20/07	7:51 AM
medium	Jun 20, 2007	7:51:30 AM
long	June 20, 2007	7:51:30 AM EDT
full	Monday, June 20, 2007	7:51:30 AM EDT

(1) SHORT 完全为数字，如 12.13.52 或 3:30pm。

(2) MEDIUM 较长，如 Jan 12, 1952。

(3) LONG 更长，如 January 12, 1952 或 3:30:32pm。

(4) FULL 是完全指定，如 Tuesday, April 12, 1952 AD 或 3:30:42pm PST。

属性 pattern 定制的格式样式可以用来对显示的数据格式作详细的指定。字母“y”表示年，字母“M”表示月，字母“d”表示日，字母“E”表示星期，字母“H”或“h”表示小时（大写表示军用，小写表示民用），字母“m”表示分钟，字母“s”表示“秒”，字母“z”表示时区。可以用这些字母进行组合，不同的字母个数或不同的组合，表示的含义也会有所不同。如“MMMM”和“MM”两者的含义是不同的。各种字母代表的含义见表 9-21 所示。

表 9-21 日期与时间模式分解

模式字符串	含 义	示 例
yy	两位数字表示的年份	07
yyyy	四位数字表示的年份	2007
M	月份	6
MM	两位数字表示的月份，不足两位在左边补 0	06
MMM	月份名称的缩写	Apr
MMMM	完整的月份名称	April
d	某月中的具体日期	5
dd	某月中的具体日期，不足两位在左边补 0	05
EE	星期几名称的缩写	Fri
EEEE	完整的星期几的名称	Friday
H	军事使用的小时	21
HH	军事使用的两位数字表示的小时，不足两位在左边补 0	21
h	小时	9
hh	两位数字表示的小时，不足两位在左边补 0	09
m	分钟	32
mm	两位数字表示的分钟，不足两位在左边补 0	32
s	秒钟	6
ss	两位数字表示的秒钟，不足两位在左边补 0	06
S	毫秒	251



续表

模式字符串	含 义	示 例
a	AM (上午) 或 PM (下午)	PM
zz	缩写表示的时区	EST
zzzz	完整的时区名称	Eastern Standard Time

### 3. <fmt:parseNumber>

<fmt:parseNumber>标签用于解析被格式化处理的表示数字、货币或百分比数据的字符串。使用<fmt:parseNumber>标签的语法如下。

第1种：没有 body

```
<fmt:parseNumber value="被解析的字符串" [type="{number|currency|percent}"]
    [pattern="定制的格式模式"] [parseLocale="区域设置"]
    [integerOnly="{true|false}"] [var="变量名"]
    [scope="{page|request|session|application}"]/>
```

第2种：带有 body

```
<fmt:parseNumber [type="{number|currency|percent}"] [pattern="定制的格式模式"]
    [parseLocale="区域设置"] [integerOnly="{true|false}"]
    [var="变量名"] [scope="{page|request|session|application}"]>
    被解析的字符串
</fmt:parseNumber>
```

<fmt:parseNumber>标签的属性说明详见表 9-22。

表 9-22 <fmt:parseNumber>标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
value	String	是	无	要被解析的字符串
type	String	否	number	被解析的字符串所代表的数值数据类型
pattern	String	否	无	被解析的字符串所使用的定制的格式模式
parseLocale	String 或 Java.util.Locale	否	无	被解析的字符串格式所使用的区域设置
integerOnly	boolean	否	false	是否放弃小数部分数据
var	String	否	无	存放解析结果的变量
scope	String	否	page	属性 var 中指定的变量的有效范围

如果属性 scope 被指定了，则属性 var 也必须被指定。指定了属性 var 则不会作输出处理，只是把解析的结果存入属性 var 指定的变量中。如果被解析的字符串的值为 null 或 empty，则不会作输出处理，即便是指定了 var 属性也会从 scope 属性指定的范围中把这个变量删除。

### 4. <fmt:parseDate>

<fmt:parseDate>标签用于解析表示日期与时间数据的字符串。使用<fmt:parseDate>标签的语法如下。

第1种：不带 body

```
<fmt:parseDate value="被解析的字符串" [type="{time|date|both}"]
```

```
[dateStyle="{default|short|medium|long|full}"]
[timeStyle="{default|short|medium|long|full}"]
[pattern="定制的格式模式"] [timeZone="时区设置"]
[parseLocale="区域设置"] [var="变量名"]
[scope="{page|request|session|application}"]/>
```

## 第2种：带有 body

```
<fmt:parseDate [type="{time|date|both}"] [dateStyle="{default|short|medium|long|full}"]
[timeStyle="{default|short|medium|long|full}"]
[pattern="定制的格式模式"] [timeZone="时区设置"]
[parseLocale="区域设置"] [var="变量名"]
[scope="{page|request|session|application}"]>
  被解析的字符串
</fmt:parseDate>
```

`<fmt:parseDate>`标签的属性说明详见表 9-23。

如果属性 `scope` 被指定了，则属性 `var` 也必须被指定。指定了属性 `var` 则不会作输出处理，只是把解析的结果存入属性 `var` 指定的变量中。如果被解析的字符串的值为 `null` 或 `empty`，则不会作输出处理，即便是指定了 `var` 属性也会从 `scope` 属性指定的范围中把这个变量删除。

表 9-23 `<fmt:parseDate>` 标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
value	String	是	无	被解析的字符串
dateStyle	String	否	default	被格式化日期的风格
timeStyle	String	否	default	被格式化时间的风格
pattern	String	否	无	被定制的格式模式
timeZone	String 或 java.util.TimeZone	否	无	数据格式的时区设置
parseLocale	String 或 Java.util.Locale	否	无	被解析的字符串格式所使用的区域设置
var	String	否	无	存放解析结果的变量，数据类型为 java..util.Date
scope	String	否	page	属性 var 中指定的变量的有效范围

## 5. `<fmt:setTimeZone>`

`<fmt:setTimeZone>` 标签用于设定时区或将设定的时区存储到一个变量中，变量的类型为 `java.util.TimeZone`。使用 `<fmt:setTimeZone>` 标签的语法如下：

```
<fmt:setTimeZone value="时区" [var="变量名"]
[scope="{page|request|session|application}"]/>
```

`<fmt:setTimeZone>` 标签的属性说明详见表 9-24。

表 9-24 `<fmt:setTimeZone>` 标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
value	String 或 java.util.TimeZone	是	无	时区设置
var	String	否	无	保存时区的变量，变量的类型为 java.util.TimeZone
scope	String	否	page	属性 var 中指定的变量的有效范围



如果 `value` 属性中的值为 `null` 或 `empty`，则时区设置为 GMT（格林威治标准时间）。

## 6. <fmt:timeZone>

**<fmt:timeZone>**标签用于设定时区。使用<fmt:timeZone>标签的语法如下：

```
<fmt:timeZone value="时区">
    body 内容（标签语句、Java 程序、HTML 代码等）
</fmt:timeZone>
```

标签本身并不处理 `body` 内容。属性 `value` 的数据类型为 `String` 或 `java.util.TimeZone` 对象。如果 `value` 属性中的值为 `null` 或 `empty`，则时区设置为 `GMT`（格林威治标准时间）。

!

■ **提示** `<fmt:timeZone>` 标签与 `<fmt:setTimeZone>` 标签的区别是：前者仅在标签嵌套的范围内有效；后者如果不是将时区存储在变量中，则会在标签之后的语句 `scope` 属性设定的范围内有效，如果是将时区存储在变量中则不会改变其他程序的时区设置。

### 【实例 9-4】使用数字与日期格式处理标签

useFormatJSTL.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<html>
<head>
    <title>使用国际化处理标签</title>
</head>
<body>
-----fmt:formatNumber 使用示例-----<br>
输出数字数据: <br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
<fmt:formatNumber value="500000.01" groupingUsed="true" />
<fmt:formatNumber value="52577" pattern="###.###E0" />
<fmt:formatNumber value="52577" minFractionDigits="4" />
<fmt:formatNumber value="52577" minIntegerDigits="4" />
<fmt:formatNumber value="52577" minIntegerDigits="1" minFractionDigits="1"/>
<br>输出货币数据: <br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
<fmt:formatNumber type="currency" value="78.74901"/>
<fmt:formatNumber type="currency" value="78.74901" currencyCode="JPY"/>
<fmt:formatNumber type="currency" value="78.74901" currencyCode="CNY"/><br>
-----fmt:formatDate 使用示例-----<br>
<jsp:useBean id="tempDate" class="java.util.Date"/>
<fmt:formatDate value="${tempDate}" type="date"/><br>
<fmt:formatDate value="${tempDate}" type="time"/><br>
<fmt:formatDate value="${tempDate}" type="both"/><br>
<fmt:formatDate value="${tempDate}"
    pattern="yyyy'-'MM'-'dd hh':'mm':ss"/><br>
<fmt:formatDate value="${tempDate}"
    pattern="yyyy'年'M月'd日'h时'm分's秒'S毫秒' a zzzz"/>
</body>
</html>
```

程序运行的结果如图 9-4 所示。

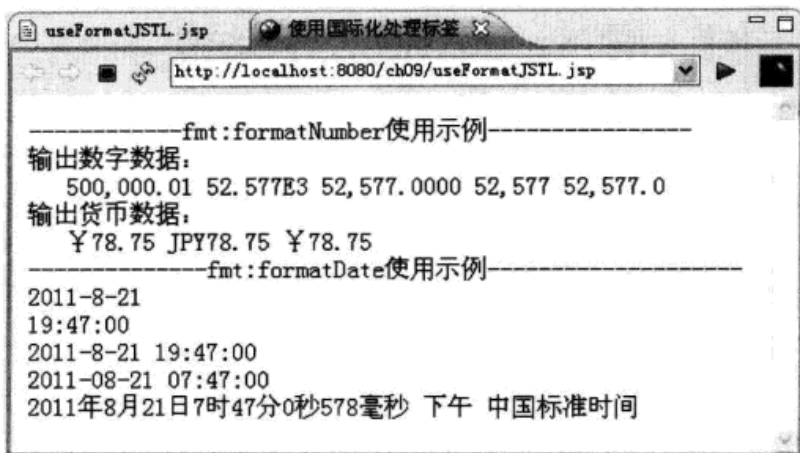


图 9-4 使用数字与日期格式处理标签

来分析程序中最难的一句：

```
<fmt:formatDate value="${tempDate}"
    pattern="yyyy'年'M'月'd'日'h'时'm'分's'秒'S'毫秒' a zzzz"/>
```

要在输出的时间表示中加入字符，以符合中国人的表示习惯，加入的字符要用两个“'”括起来；随后的“a”表示根据时间来输出是上午还是下午；“zzzz”表示输出所在的时区。

## 9.4 SQL 标签

SQL 标签使用方便，开发迅速，编写出来的代码可读性良好，但也有许多开发人员反对使用 SQL 标签，其根本原因是认为使用 SQL 标签破坏了软件的体系架构，比如 MVC 架构中，将 M（Model，模型）中的功能也放在 V（View）中直接实现了；其他就是安全性不够好，如果一个 JSP 页面出错或被攻击者得到源代码，则可能连接数据库的用户名和密码、操作数据的 SQL 语句都将暴露开来。但是 SQL 标签在一些应用开发的场合仍是有较大的使用价值，比如要快速开发一个系统的原型、要开发一个小型的管理信息系统等。

SQL 标签主要有 6 个：<sql:setDataSource>标签、<sql:query>标签、<sql:param>标签、<sql:update>标签、<sql:dateParam>标签、<sql:transaction>标签，用于设置数据源，做数据查询操作（select），做数据更新操作（update、insert、delete），以及做事务处理。要使用 SQL 标签需要在 JSP 文件首部加入如下语句：

```
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
```

### 9.4.1 设置数据源

可以使用<sql:setDataSource>标签来设置一个数据源。使用<sql:setDataSource>标签的语法如下：

```
<sql:setDataSource {dataSource="数据源" |url="JDBC 连接 URL"
    [driver="驱动程序类名"] [user="连接数据库时使用的用户名"]
    [password="连接数据库的用户的密码"]} [var="变量名"]
    [scope="{page|request|session|application}"/>
```



如果属性 `dataSource` 的值为 `null`，将抛出一个 `JspException` 异常。属性 `dataSource` 的值可以是一个已定义好的 `dataSource` 对象、JNDI 路径、JDBC 连接参数字符串，也可以通过 4 个属性 `url`、`driver`、`user`、`password` 的配合来连接数据库。`<sql:setDataSource>` 标签的属性说明详见表 9-25。

表 9-25 `<sql:setDataSource>` 标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
<code>dataSource</code>	String 或 <code>javax.sql.DataSource</code>	否	无	数据源
<code>url</code>	String	否	无	连接数据库的 URL
<code>driver</code>	String	否	无	连接数据库的驱动程序类名
<code>user</code>	String	否	无	连接数据库时使用的用户名
<code>password</code>	String	否	无	连接数据库时使用的用户密码
<code>var</code>	String	否	无	代表数据源的变量名
<code>scope</code>	String	否	无	<code>var</code> 属性指定的变量的有效范围

**提示** 属性 `dataSource` 的设置推荐使用 JNDI 路径，因为这样就不必在每个 JSP 页面都出现连接数据库的参数了，如果连接参数有变化只要修改相关的配置文件就可以了，而不必修改每个 JSP 页面中的程序。

**提示** 由于这里我们使用的是 JDK 7，因此 `driverName` 参数在编程时不必给出，JDK 7 会根据 `url` 自动加载驱动程序。

## 9.4.2 查询数据

`<sql:query>` 标签用于查询数据库中的数据，这就相当于执行 `select` 查询 SQL 语句。使用 `<sql:query>` 标签的语法如下。

第 1 种：不带 `body`

```
<sql:query sql="SQL 查询语句"
    var="变量名" [scope="{page|request|session|application}"]
    [dataSource="数据源"] [maxRows="返回的记录集最大行数"]
    [startRow="返回的记录集的起始位置"] />
```

第 2 种：带有指定查询参数的 `body`

```
<sql:query sql=" SQL 查询语句"
    var="变量名" [scope="{page|request|session|application}"]
    [dataSource="数据源"] [maxRows="返回的记录集最大行数"]
    [startRow="返回的记录集的起始位置">
    <sql:param> 标签语句
</sql:query>
```

## 第 3 种：带有指定查询和查询参数选项的 body

```

<sql:query var="变量名" [scope="{page|request|session|application}"]
    [dataSource="数据源"] [maxRows="返回的记录集最大行数"]
    [startRow="返回的记录集的起始位置">
    SQL 查询语句
    <sql:param> 标签语句
</sql:query>

```

第 1 种语法最简单,第 2 种和第 3 种语法就像是使用了 JDBC 中的 PreparedStatement 对象。  
<sql:query>标签的属性说明详见表 9-26。

表 9-26 &lt;sql:query&gt;标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
sql	String	是	无	查询数据的 SQL 语句
dataSource	String 或 javax.sql.DataSource	否	无	数据源,可以是一个已定义好的 dataSource 对象、JNDI 路径、JDBC 连接参数字符串
maxRows	int	否	无	返回查询结果记录集的最大行数
startRow	int	否	无	返回查询结果记录集的起始位置
var	String	否	无	保存查询结果的对象,数据类型为 javax.servlet.jsp.jstl.sql.Result
scope	String	否	page	var 属性指定变量的有效范围

如果没有设置属性 maxRows 或设置为“-1”,则表示对最大行数不作限制;startRow 是指返回的记录集在 SQL 查询结果中的起始位置,首行的位置为“0”,如果没有指定此属性的值,则默认为“0”。从 SQL 查询结果记录集的位置“0”到“startRow-1”的记录将不会被包含到<sql:query>标签查询得到的结果记录集对象中。属性 maxRows 设置的值必须大于等于-1。

<sql:query>标签从数据库中查询数据将结果数据集放入到 var 属性指定的变量中。如果找到需要查询的数据,则得到一个结果为 empty 的 javax.servlet.jsp.jstl.sql.Result 对象,即记录条数为 0。

startRow 或 maxRows 这两个属性在做数据分页处理时特别有用,startRow 可用于指定当前页在 SQL 查询结果中的起始位置,maxRows 用于指定页面的尺寸,即当前页面的记录条数。



**提示** <sql:query>标签的分页处理并不能从根本上减少网络流量,如果要降低流量,应当想办法构造出仅查询当前页数据的 SQL 语句。

查询数据的 SQL 语句,如果使用的是第 1 种语法则在属性 sql 中设定,如果使用的是第 2 种语法和第 3 种语法则在标签的 body 内容中指定。SQL 语句如果是参数语句,即其中含有“?”的语句,这就需要有<sql:param>标签语句来设置其中的“?”参数的值。

<sql:param>标签用于设置数据库操作标签语句中 SQL 语句中的参数值,嵌套<sql:param>的标签可以是<sql:query>、<sql:update>。

```
<sql:param value="参数值"/>
```

或

```

<sql:param>
    参数值
</sql:param>

```

如果参数值设置为 null,则会将 SQL 语句中相应的“?”参数的值设为“NULL”。



### 9.4.3 显示查询结果

通过`<sql:query>`标签可以得到查询的结果记录集，要显示数据就需要用到这个结果记录集的一些属性，主要有如下几个。

(1) `rows`: 返回的是一个 `java.util.SortedMap` 对象一维数组，数组中的每一个元素对应查询结果记录集中特定的一行。

(2) `rowsByIndex`: 返回的是一个 `java.lang.Object` 二维数组，第一维代表结果记录集中特定的一行。

(3) `columnNames`: 字段名（即列名）`String` 数组。

(4) `rowCount`: 结果记录集中记录的总条数。

(5) `limitedByMaxRows`: 标签中记录条数最多的属性值。

因此可根据结果记录集再结合以上的 5 个属性，使用`<c:forEach>`标签循环输出结果记录集中的数据。

#### 【实例 9-5】运用 SQL 标签查询数据

为调试本节中的程序，首先在 `testDatabase` 数据库中创建一个表：`account`。执行建表的 SQL 语句，语句如下：

```
--如果已存在 account 表则删除此表
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[account]') and
    OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[account]
GO
--创建 account 表
CREATE TABLE [dbo].[account] (
    [account_id] [bigint] IDENTITY (1, 1) NOT NULL ,
    [account_name] [varchar] (80) COLLATE Chinese_PRC_CI_AS NULL ,
    [account_money] [money] NULL ,
    [add_time] [datetime] NULL
) ON [PRIMARY]
GO
--声明 add_time 字段的默认值为 getdate(), account 表的主键为 account_id
ALTER TABLE [dbo].[account] ADD
    CONSTRAINT [DF_account_add_time] DEFAULT (getdate()) FOR [add_time],
    CONSTRAINT [PK_account] PRIMARY KEY CLUSTERED
(
    [account_id]
) ON [PRIMARY]
GO
```

`account_id` 字段表示账户 ID 号，即账号，并设置为 `bigint` 类型，数据自动增量，步长值为 1，设为主键，以唯一标识账户；`account_name` 字段表示账户名，即客户的名称；`account_money`

字段表示账户的余额；add\_time 字段表示账户创建的时间，默认值为 getdate()，即创建记录时的时间。

**提示** 实际工程中，银行的账户表设计会更复杂一些，比如账号的生成规则、账户表的信息字段数等，而且很少有银行采用 SQL Server 数据库，大多采用 Unix 系统下的数据库数据，但学习 JSP 编程并不必太多关心数据库的品种，不同的数据库品种使用的数据库驱动程序不同，操作数据库时使用的 SQL 语句也会稍有差异，因此读者在编程时应尽量使用标准 SQL 语句以方便程序的移植。

建表成功后，随机输入一些数据，以方便调试程序，最好能输入 10 行以上的数据，以方便调试本章后续的分页数据处理等程序实例。输入数据的方法可以在企业管理器中打开表直接输入数据，也可以在查询分析器中执行 insert SQL 语句，但编写 SQL 语句时应注意到 account\_id、add\_time 字段的值可以自动生成，不必给出值，如：

```
insert into account(account_name,account_money) values('邓佳容',1000.00)
```

#### viewData.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<html>
<head>
    <title>使用 SQL 标签</title>
</head>
<body>
    -----sql:query 标签与 sql:param 标签应用示例-----<br>
    使用 JNDI 方式建立数据源：
    <sql:setDataSource
        url="jdbc:sqlserver://localhost:1433;DatabaseName=testDatabase"
        user="sa" password="123" var="sqlDS"/>
    <c:if test="${sqlDS==null}">
        建立数据源失败！
    </c:if>
    <c:if test="${sqlDS!=null}">
        建立数据源成功！
    </c:if>
    <sql:query
        sql="select top 5 * from account"
        var="accountsRS" dataSource="${sqlDS}"/>
    共查询出<c:out value="${accountsRS.rowCount}"/>条记录<br>
    <hr>输出查询结果（使用 rows 输出）：
    <table border="1" cellpadding="0" cellspacing="0">
        <tr>
            <c:forEach items="${accountsRS.columnNames}" var="cucN">
                <td><c:out value="${cucN}"/></td>
            </c:forEach>
        </tr>
        <c:forEach items="${accountsRS.rows}" var="cuRe">
```



```

        <tr>
            <c:forEach items="${cuRe}" var="cuValue" varStatus="currentVarStaus">
                <td><c:out value="${cuRe[accountsRS.columnNames[currentVarStaus.index]]}"/></td>
            </c:forEach>
        </tr>
    </c:forEach>
</table>
<hr>输出查询结果（使用 rowsByIndex 输出）:
<table border="1" cellpadding="0" cellspacing="0">
    <tr>
        <c:forEach items="${accountsRS.columnNames}" var="cucN">
            <td><c:out value="${cucN}"/></td>
        </c:forEach>
    </tr>
    <c:forEach items="${accountsRS.rowsByIndex}" var="cuRe">
        <tr>
            <c:forEach items="${cuRe}" var="cuValue">
                <td><c:out value="${cuValue}"/></td>
            </c:forEach>
        </tr>
    </c:forEach>
</table>
</body>
</html>

```

程序的运行结果如图 9-5 所示。



图 9-5 查询数据并显示

显示查询结果有两种方法：一种是使用 rows 记录集，一种是使用 rowsByIndex 记录集。

(1) 使用 rows 输出查询结果。

rows 记录集是一个 java.util.SortedMap 对象一维数组，数组中的每一个元素对应查询结果

记录集中特定的一行。输出 `java.util.SortedMap` 对象中的数据时，本例使用了如下的语句：

```
<c:forEach items="${ accountsRS.rows}" var="cuRe">
  <tr>
    <c:forEach items="${cuRe}" var="cuValue" varStatus= "currentVarStaus">
      <td><c:out value="${cuRe[accountsRS.columnNames[currentVarStaus.index]]}"/></td>
    </c:forEach>
  </tr>
</c:forEach>
```

外层的 `<c:forEach>` 标签遍历的是 `java.util.SortedMap` 对象一维数组，内层的 `<c:forEach>` 标签遍历的是每一个 `java.util.SortedMap` 对象中的元素。输出元素时不能使用如下的语句：

```
<c:out value="${cuValue}"/>
```

这是因为 `java.util.SortedMap` 对象并不能保证其中的元素是有序的，更不能保证顺序与输出列名的顺序相同，因此在输出值时，要多一些约束条件。此处，输出方式为：

```
<c:out value="${cuRe[accountsRS.columnNames[currentVarStaus.index]]}"/>
```

`currentVarStaus.index` 表示当前输出的 `java.util.SortedMap` 对象中的元素的下标，`accountsRS.columnNames[currentVarStaus.index]` 得到的是对应的结果记录集中的列名，因此，`cuRe[accountsRS.columnNames[currentVarStaus.index]]` 得到的就是与 `accountsRS.columnNames` 对象顺序对应起来的值。这样做最根本的思想是考虑到 `accountsRS.columnNames` 对象的元素个数与 `java.util.SortedMap` 对象的元素个数相同。

可见，内层的 `<c:forEach>` 标签内输出的并不一定是当前遍历的 `java.util.SortedMap` 对象中的元素值，而是 `java.util.SortedMap` 对象中与 `accountsRS.columnNames` 对象中相对应的元素值。

(2) 使用 `rowsByIndex` 输出查询结果。

`rowsByIndex` 是一个 `java.lang.Object` 二维数组，第一维代表结果记录集中特定的一行，因此输出某条记录的某个字段的值时需要双下标，或使用嵌套的 `<c:forEach>`，如下所示：

```
<c:forEach items="${ accountsRS.rowsByIndex}" var="cuRe">
  <tr>
    <c:forEach items="${cuRe}" var="cuValue">
      <td><c:out value="${cuValue}"/></td>
    </c:forEach>
  </tr>
</c:forEach>
```

幸好 `rowsByIndex` 中元素出现的顺序与 `columnNames` 中的顺序是相同的，故不必再作元素定位即可以直接输出。

## 9.4.4 更新数据

对数据库中数据的 `insert`、`update`、`delete` 等更新操作 SQL 语句都可以通过 `<sql:update>` 标签来执行。`<sql:update>` 标签也可以用来执行没有返回结果的 SQL DDL (Data Definition Language, 数据定义语言) 语句，但实践工程中很少这样做。使用 `<sql:update>` 标签的语法如下。

第 1 种：不带 body

```
<sql:update sql="SQL 语句" [dataSource="数据源"]
  [var="变量名"] [scope="{page|request|session|application}"/>
```



第 2 种：带有指定 SQL 语句参数的 body

```
<sql:update sql="SQL 语句" [dataSource="数据源"]
    [var="变量名"] [scope="{page|request|session|application}"]>
    <sql:param>标签语句
</sql:update>
```

第 3 种：带有指定的 SQL 语句和语句参数的 body

```
<sql:update [dataSource="数据源"] [var="变量名"]
    [scope="{page|request|session|application}"]>
    SQL 语句
    <sql:param>标签语句
</sql:update>
```

<sql:update>标签的属性说明详见表 9-27。

表 9-27 <sql:update>标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
sql	String	是	无	操作数据的 SQL 语句
dataSource	String 或 javax.sql.DataSource	否	无	数据源
var	String	否	无	代表操作结果的变量,数据类型为 java.lang.Integer
scope	String	否	无	var 属性指定的变量的有效范围

如果属性 scope 被指定了,则属性 var 必须给定值。如果属性 dataSource 被指定,即指定了数据源,则<sql:update>标签不能被嵌套在<sql:transaction>标签语句中。如果属性 dataSource 的值为 null,将抛出 JspException 异常。

属性 var 指定的变量存放的是 SQL 语句(insert、update、delete)操作所影响的记录条数,如果值为 0 表示没有数据被更新。如果是 SQL DDL 语句,则没有返回值。这就相当于执行了 JDBC 中的 executeUpdate()方法。

9.4.5 日期型数据处理

<sql:dateParam>标签用于设置 SQL 语句中的 java.util.Date 类型参数的值。使用<sql:dateParam>标签的语法如下:

```
<sql:dateParam value="值" [type="{timestamp|time|date}"]/>
```

<sql:dateParam>标签的属性说明详见表 9-28。

表 9-28 <sql:dateParam>标签的属性

属 性	数据类型	是否必选项	默 认 值	属性值的说明
value	java.util.Date	是	无	操作数据库表的 SQL 语句中的参数值,在数据库表中的字段类型为 date、time 或 timestamp
type	String	否	timestamp	数据库中字段的数据类型

`<sql:dateParam>` 标签需要嵌套在 `<sql:query>` 标签、`<sql:update>` 标签中使用。如果属性 `value` 的值为 `null`，则 SQL 语句中该参数的值会被设为“NULL”。

`<sql:dateParam>` 标签会根据标签中属性的设置情况和实际需要，将 `java.util.Date` 类型数据转换成 SQL 数据库中所需的 `java.sql.Date`、`java.sql.Time` 和 `java.sql.Timestamp` 类型数据。

### 9.4.6 事务处理

`<sql:transaction>` 标签用于为 `<sql:query>` 标签和 `<sql:update>` 标签做事务处理。使用 `<sql:transaction>` 标签的语法如下：

```
<sql:transaction [dataSource="数据源"]
    [isolation="事务隔离级别"]>
    <sql:query>标签和<sql:update>标签语句
</sql:transaction>
```

“事务隔离级别”为以下 4 个中的 1 个：`read_committed`、`read_uncommitted`、`repeatable_read`、`serializable`。`read_uncommitted` 表示一个事务在提交前其变化对于其他事务来说是可见的，在这种级别下，脏读、不可重复的读、虚读都是允许的。`read_committed` 表示读取未提交的数据是不允许的，这个级别下仍然允许不可重复的读和虚读。`repeatable_read` 保证能够再次读取相同的数据而不会失败，但虚读仍然可能会出现。`Serializable` 是最高的事务级别，防止脏读、不可重复的读和虚读。

如果属性 `dataSource` 的值为 `null`，将抛出 `JspException` 异常。事务中的 SQL 语句执行过程中如果发生异常，需要捕获异常并作出处理，事务会自动回滚。

#### 【实例 9-6】运用 SQL 标签做事务处理

假设现在要从账号为“2”的账户中转账 10 元到账号为“1”的账户中，要实现这个功能，就需要作事务处理。来看程序代码。

transcation.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<html>
<head>
    <title>使用 SQL 标签</title>
</head>
<body>
    -----事务前的数据-----<br>
    <sql:setDataSource
        url="jdbc:sqlserver://localhost:1433;DatabaseName=testDatabase"
        user="sa" password="123" var="sqlDS"/>
    <sql:query
        sql="select top 5 * from account"
        var="accountsRS" dataSource="${sqlDS}"/>
    共查询出<c:out value="${accountsRS.rowCount}"/>条记录<br>
```



```

<hr>输出查询结果（使用 rows 输出）：
<table border="1" cellpadding="0" cellspacing="0">
  <tr>
    <c:forEach items="${accountsRS.columnNames}" var="cucN">
      <td><c:out value="${cucN}"/></td>
    </c:forEach>
  </tr>
  <c:forEach items="${accountsRS.rows}" var="cuRe">
    <tr>
      <c:forEach items="${cuRe}" var="cuValue" varStatus="currentVarStaus">
        <td><c:out value="${cuRe[accountsRS.columnNames[
          currentVarStaus.index]]}"/></td>
      </c:forEach>
    </tr>
  </c:forEach>
</table>
-----进行事务处理-----<br>
<c:catch var="error">
  <sql:transaction dataSource="${sqlDS}">
    <sql:update
      sql="update account set account_money=account_money+? where account_id=?">
      <sql:param value="${10}"/>
      <sql:param value="1"/>
    </sql:update>
    <sql:update
      sql="update account set account_oney=account_money-? where account_id=?">
      <sql:param value="${10}"/>
      <sql:param value="2"/>
    </sql:update>
  </sql:transaction>
</c:catch>
出现异常：<c:out value="${error}"/><br>
-----事务后的数据-----<br>
<sql:query
  sql="select top 5 * from account"
  var="accountsRS" dataSource="${sqlDS}"/>
共查询出<c:out value="${accountsRS.rowCount}"/>条记录<br>
<hr>输出查询结果（使用 rows 输出）：
<table border="1" cellpadding="0" cellspacing="0">
  <tr>
    <c:forEach items="${accountsRS.columnNames}" var="cucN">
      <td><c:out value="${cucN}"/></td>
    </c:forEach>
  </tr>
  <c:forEach items="${accountsRS.rows}" var="cuRe">
    <tr>
      <c:forEach items="${cuRe}" var="cuValue" varStatus="currentVarStaus">
        <td><c:out value="${cuRe[accountsRS.columnNames[
          currentVarStaus.index]]}"/></td>
      </c:forEach>
    </tr>
  </c:forEach>
</table>
</body>
</html>

```

程序运行结果如图 9-6 所示。



图 9-6 运用 SQL 标签作事务处理

从图中的数据来看,事务后转账并没有成功,也就是说事务失败了,程序自动作了回滚。程序中通过<c:catch>标签捕获了异常,根据异常可以得知错误的原因是在第2个更新操作的SQL语句中,account\_money 字段名写错了,写成了 account\_oney,这种写错代码的情况是比较常见的。



**提示** 在设置 SQL 语句中的参数时,像 money 这种字段类型,用<sql:param>时,value 属性中需要使用 EL 表达式,否则数据库将 value 属性中的值当作字符串处理,但又无法将字符串自动转换为 money 类型的数据。如果使用了 EL 表达式,如程序代码中所示,用的是\${10},则会将 10 当作数值型数据,从而可以成功地转换成 money 类型的数据。如果采用如下的语句:

```
<sql:param value="10"/>
```

则会抛出如下所示的异常:

```
javax.servlet.jsp.JspException: update account set account_money=account_money-? where account_id=? : [Microsoft][SQLServer 2000 Driver for JDBC] [SQLServer] 不允许从数据类型 nvarchar 到数据类型 money 的隐性转换 (表 'testDatabase.dbo.account', 列 'account_money')。请使用 CONVERT 函数来运行此查询。
```

## 9.5 函数标签

函数标签用在 EL 表达式中,用于字符串处理,诸如大小写转换、查找子串、截取子串之类的操作。根据操作功能的不同,可以将函数标签分为:大小写转换函数、求子串函数、去空



白函数、替换函数、查找函数、拆分与组合函数、XML 符号转换函数、求长度函数。

如果要使用函数标签库中的函数，需要在 JSP 页面的首部加入如下的语句：

```
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

### 9.5.1 求长度函数

`length` 函数用于求某个对象中元素的个数。使用函数 `length` 的语法如下：

```
fn:length(要求元素个数的对象)
```

此函数返回 `int` 型数据类型。要求元素个数的对象数据类型是可以用 `<c:forEach>` 作迭代处理的数据类型，也可以是一个字符串，如果是字符串则函数返回字符串中字符的个数。如果要求元素个数的对象为 `null`，则函数返回 0。



**提示** 求长度的对象并不限于字符串，只要是能用 `<c:forEach>` 作迭代处理的数据类型就可以。也就是说，可以是实现了 `java.util.Collection` 接口的类，包括 `List`、`LinkedList`、`ArrayList`、`Vector`、`Stack`、`Set`；可以是实现了 `java.util.Map` 接口的类，包括 `HashMap`、`Hashtable`、`Properties`、`Provider`、`Attributes`；可以是数组 `arrays`，数组中的数据是简单数据类型（如：`int`、`float` 等）；可以是实现了 `java.util.Iterator` 和 `java.util Enumeration` 的类；可以是一个用逗号分隔的字符串（如：“Monday,Sunday,Tuesday”）。

### 9.5.2 大小写转换函数

#### 1. toLowerCase

`toLowerCase` 函数用于将字符串中的字母全部转换为小写，函数返回转换后的字符串，函数的功能相当于调用 `java.lang.String` 类的 `toLowerCase()` 方法。使用 `toLowerCase` 函数的语法如下：

```
fn:toLowerCase(要转换的字符串)
```

#### 2. toUpperCase

`toUpperCase` 函数用于将字符串中的字母全部转换为大写，函数返回转换后的字符串，函数的功能相当于调用 `java.lang.String` 类的 `toUpperCase()` 方法。使用 `toUpperCase` 函数的语法如下：

```
fn:toUpperCase(要转换的字符串)
```

### 9.5.3 求子串函数

#### 1. substring

`substring` 函数用于求一个字符串的子字符串，功能与 `java.lang.String` 的 `substring()` 方法类似。使用函数 `substring` 的语法如下：

```
fn:substring(母串,开始下标,结束下标)
```

如果开始下标比结束下标还大，则返回结果为空串。如果开始下标小于 0，则开始下标会

被调整为 0。如果结束下标小于 0 或大于母串的长度，结束下标会被调整为母串的长度。如果开始下标比母串的最大下标还要大，则返回结果为空串。



**提示** 字符串的下标从 0 开始，结果字符串中并不包括结束下标处的字符。

## 2. substringAfter

substringAfter 函数用于求一个母串中指定的子字符串（简称子串）之后的字符串。使用函数 substringAfter 的语法如下：

```
fn:substringAfter(母串,子串)
```

## 3. substringBefore

substringBefore 函数用于求一个母串中指定的子串之前的字符串。使用函数 substringBefore 的语法如下：

```
fn:substringBefore(母串,子串)
```

## 9.5.4 去空白函数

trim 函数用于去掉字符串的左右空白字符，返回去掉空白字符后的字符串。使用函数 trim 的语法如下：

```
fn:trim(要去掉左右空白字符的字符串)
```

空白字符有：[\t\n\x0B\f\r]



**提示** 函数 trim 只是去掉字符串的左右空白字符，并不会去掉字符串中间的空白字符。

## 9.5.5 替换函数

replace 函数用于将母字符串中的某个子字符串替换成指定的字符串，返回替换后的结果母字符串。使用函数 replace 的语法如下：

```
fn:replace(母字符串, 要替换的子字符串, 替换成的字符串)
```

## 9.5.6 查找函数

### 1. indexOf

indexOf 函数得到的字符串是指定的子字符串第一次出现的下标。使用函数 indexOf 的语法如下：

```
fn:indexOf(字符串,子字符串)
```

如果子字符串为 null，则将其当作空串处理，函数返回的结果为 0。



## 2. startsWith

函数 `startsWith` 判断一个字符串是否以另一个字符串开头，函数的返回结果是一个布尔值。使用函数 `startsWith` 的语法如下：

```
fn:startsWith(字符串 1, 字符串 2)
```

函数 `startsWith` 判断字符串 1 是否以字符串 2 开头，如果是返回 `true`，否则返回 `false`。如果字符串 2 为 `null`，则将字符串 2 当作空串处理，函数返回 `true`。

## 3. endsWith

函数 `endsWith` 判断一个字符串是否以另一个字符串结尾，函数的返回结果是一个布尔值。使用函数 `endsWith` 的语法如下：

```
fn:endsWith(字符串 1, 字符串 2)
```

函数 `endsWith` 的功能就是判断字符串 1 是否以字符串 2 结尾，如果是返回 `true`，否则返回 `false`。如果字符串 2 为 `null`，则将字符串 2 当作空串处理，函数返回 `true`。

## 4. contains

函数 `contains` 判断一个字符串中是否包含另一个字符串，函数的返回结果是一个布尔值。使用函数 `contains` 的语法如下：

```
fn:contains (字符串 1, 字符串 2)
```

函数 `contains` 的功能就是判断字符串 1 是否包含字符串 2，如果是返回 `true`，否则返回 `false`。如果字符串 2 为 `null`，则将字符串 2 当作空串处理，函数返回 `true`。

## 5. containsIgre

函数 `containsIgre` 在不区分大小写的情形下判断一个字符串中是否包含另一个字符串，函数的返回结果是一个布尔值。使用函数 `containsIgre` 的语法如下：

```
fn:containsIgre (字符串 1, 字符串 2)
```

函数 `containsIgre` 的功能就是在不区分大小写的情形下判断字符串 1 是否包含字符串 2，如果是返回 `true`，否则返回 `false`。如果字符串 2 为 `null`，则将字符串 2 当作空串处理，函数返回 `true`。

# 9.5.7 拆分与组合函数

## 1. split

函数 `split` 用于将一个字符串拆分成一个字符串数组。使用函数 `split` 的语法如下：

```
fn:split(字符串, 拆分的分界字符串)
```

如果拆分的分界字符串在字符串中找不到，则返回一个只有一个元素的数组，其中的这个元素就是要拆分的字符串。此外，拆分后的数组元素字符串中并不包括拆分的分界字符串。

## 2. join

函数 `join` 用于将一个字符串数组中的所有元素组合成一个字符串，返回组合后的字符串。

使用函数 join 的语法如下：

```
fn:join(字符串数组,组合分界字符串)
```

## 9.5.8 XML 符号转换函数

函数 escapeXml 用于将字符串中的 XML 标记转换为实体字符，返回转换后的结果字符串。使用函数 escapeXml 的语法如下：

```
fn:escapeXml(要转换的字符串)
```

### 【实例 9-7】函数标签应用示例

usefn.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<html>
  <head>
    <title>函数使用示例</title>
  </head>
  <body>
    -----length 函数使用示例-----<br>
    <c:set var="str" value="电子工业出版社(PHEI)"/>
    字符串"电子工业出版社(PHEI)"的长度为: ${fn:length(str)}<br>
    -----字符串大小写转换函数使用示例-----<br>
    字符串"电子工业出版社(PHEI)"转换为小写: ${fn:toLowerCase(str)}<br>
    -----求子串函数使用示例-----<br>
    字符串"电子工业出版社(PHEI)"的下标 0 到下标 4 的子串是.
    ${fn:substring(str,0,4)}<br>
    -----replace 函数使用示例-----<br>
    将字符串"电子工业出版社(PHEI)"中的 "(" 替换为 ":" :
    <c:set var="frontStr" value="("/>
    <c:set var="backStr" value=":"/>
    ${fn:replace(str,frontStr,backStr)}<br>
  </body>
</html>
```

程序的运行结果如图 9-7 所示。

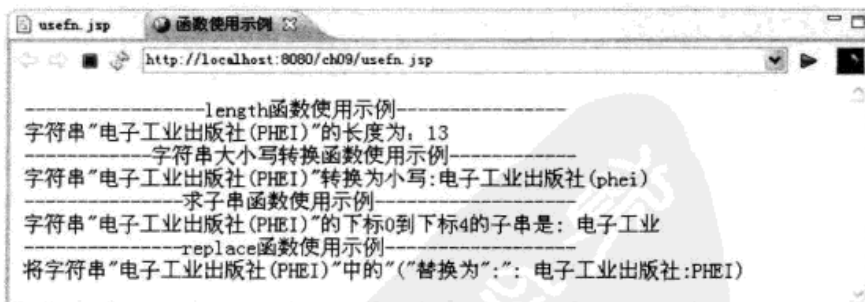


图 9-7 函数标签应用示例

从程序和运行的结果可以看出，字符串中一个汉字的长度为 1；字符串的下标是从 0 开始



的，使用 `substring` 函数时，结果并不包括结束下标位置的字符。

## 9.6 小结

JSTL 简化了 JSP 和 Web 程序的开发，统一了开发接口，能方便地在各种服务器之间进行移植。JSTL 的标签按功能分类可以分为：核心标签库、XML 处理标签库（工程中较少使用）、国际化处理标签库、SQL 标签库、函数标签库等。当要使用不同的标签库中的 JSTL 标签时，需要在 JSP 页面的首部加入如下的语句：

```
<%@ taglib prefix="标签库使用时的前缀" uri="标签库的 URI" %>
```

核心标签库中有最为常用的一些标签，包括 `<c:out>`、`<c:set>`、`<c:remove>`、`<c:catch>`、`<c:if>`、`<c:choose>`、`<c:when>`、`<c:otherwise>`、`<c:forEach>`、`<c:forTokens>`、`<c:import>`、`<c:url>`、`<c:redirect>` 等。

国际化格式标签能让数据适应不同的国家、语言、文化要求，分为国际化类、消息类、数字日期格式化类 3 种 24 个，包括有 `<fmt:setLocale>`、`<fmt:requestEncoding>`、`<fmt:bundle>`、`<fmt:message>`、`<fmt:setBundle>`、`<fmt:param>`、`<fmt:formatNumber>`、`<fmt:formatDate>`、`<fmt:parseDate>`、`<fmt:parseNumber>`、`<fmt:setTimeZone>`、`<fmt:timeZone>` 等。

SQL 标签主要有 6 个：`<sql:setDataSource>` 标签、`<sql:query>` 标签、`<sql:param>` 标签、`<sql:update>` 标签、`<sql:dateParam>` 标签、`<sql:transaction>` 标签，用于设置数据源，做数据查询操作（select），做数据更新操作（update、insert、delete），以及做事务处理。

函数标签用在 EL 表达式中，用于字符串处理，诸如大小写转换、查找子串、截取子串之类的操作。根据操作功能的不同，可以将函数标签分为：大小写转换函数、求子串函数、去空白函数、替换函数、查找函数、拆分与组合函数、XML 符号转换函数、求长度函数。

## 9.7 练习

1. 现有一个从其他信息系统发来的报文（可以理解为字符串，异构环境下的信息系统的数据交换经常用到），报文内容如下：

```
username=testusr|password=abcde|accountno=80034334324434234|
moneynumber=10000.00
```

说明：username：用户名，password：交易密码，accountno：账号，moneynumber：交易金额。

请编写一个 JSP 程序，声明一个 String 变量，变量的值为上面的报文，试用 `<c:forTokens>` 标签处理报文并显示处理结果。

2. 使用 SQL 处理标签将 userTable 表中的用户年龄增 1，并结合核心标签显示修改后的 userTable 表中的内容。

# 10

## EL 表达式

---

在 JSP2.0 及其以后的版本中引入了 EL (Expression Language, 表达式语言), 它既可以和 JSP 中的 Java 代码结合使用, 也可以和 JSTL 结合使用, 可以减少 JSP 页面中的 Java 代码量。

本章将介绍 EL 表达式的语法、隐含对象及各种运算符, 并辅以实例说明。通过本章的学习, 读者应当能在 JSP 页面中灵活运用 EL, 并结合其他代码完成所需的功能。

### 10.1 EL 简介

在 JSP 2.0 及其以后的版本中引入了 EL, Tomcat 7 中可以直接使用 EL。EL 具有如下特点。

- (1) 可得到 PageContext 属性值。
- (2) 可直接访问 JSP 的隐含对象, 如 request、session、application、page 等。
- (3) 运算符丰富, 有关系运算符、逻辑运算符和算术运算符等。
- (4) 扩展函数可与 Java 类的静态方法对应起来, 可用于自定义函数, 以增强 EL 表达式的功能 (此部分内容在后续章节中再作详细说明)。
- (5) 具有丰富的可用于 EL 表达式的函数, 可大大降低 Java 代码编写的复杂程度 (此部分内容在后续章节中再作详细说明)。

#### 10.1.1 运算符

EL 表达式的运算符特别丰富, 使用起来也相当方便。使用 EL 表达式采用如下的简单书写方法:

`${表达式}`

##### 1. “.” 与 “[ ]”

EL 提供了 “.” 和 “[ ]” 两个运算符来存取数据, 其功能含义大致相同, 但使用时有细微的差别。



假设有一个 session 变量 user 为 JavaBean 组件，其有一属性 name，表示用户的名字。要取出变量 user 属性 name 的值，EL 表达式可为：

```
${sessionScope.user.name} 或 ${sessionScope.user["name"]}
```

两者的功能相同，但如下的情况请使用 “[]” 运算符：

(1) 当要存取的属性名称中包含一些特殊字符时，就一定要使用 “[]”，如：含有 “.” 或 “-” 等非字母及数字的符号，EL 表达式为：

```
${sessionScope.user["my-name"]}
```

(2) 当需要动态改变所要存取的属性名称时，就要使用 “[]”，如：nameVariable 变量的值为属性名称字符串，得到 nameVariable 的值是所代表的属性名称时，EL 表达式为：

```
${sessionScope.user[nameVariable]}
```

而不能写成：

```
${sessionScope.user.nameVariable}
```

那将是取出 user 的 nameVariable 属性，而不是将 nameVariable 的值作为属性名。

此外使用 “[]” 运算符时，有如下的表达式：

```
${表达式 A[表达式 B]}
```

- a. 当表达式 A 值为 null，则整个表达式结果为 null；
- b. 当表达式 B 值为 null，则整个表达式结果为 null；
- c. 当表达式 A 为一个 Map 类型的数据时，如果此 Map 类型数据中并不包含表达式 B，则整个表达式结果为 null，否则整个表达式结果为此 Map 类型数据中的表达式 B 的值。
- d. 如果表达式 A 为一个 List 或 array 类型数据时，将把表达式 B 的值强制转换为 int 型数据，如果转换不成功会报错，如果下标溢出会抛出 `ArrayIndexOutOfBoundsException` 异常或 `IndexOutOfBoundsException` 异常，如果没有出现错误则会得到相应元素的值。
- e. 如果表达式 A 为一个 JavaBean 对象，则表达式 B 的值会被强制转换为一个 String 类型数据，并返回相应属性的值。

## 2. 算术运算符

EL 表达式的算术运算符主要有 5 个：+（加）、-（减）、\*（乘）、/或 div（除）、%或 mod（模运算，即取余数），如表 10-1 所示。

表 10-1 算术运算符

算术运算符	说 明	示 例	示例结果
+	加	<code>\${2+3}</code>	5
-	减	<code>\${2-3}</code>	-1
*	乘	<code>\${2*3}</code>	6
/或 div	除	<code>\${2/3}</code> 或 <code>\${2 div 3}</code>	0.6666666666666666
%或 mod	取余	<code>\${2%3}</code> 或 <code>\${2 mod 3}</code>	2



**提示** `{2/3}` 的运算结果为小数，如果需要显示为整数可使用 `fmt` 标签。在使用英文的运算符时，运算数与运算符之间应当有一个空格。

### 3. 关系运算符

EL 的关系运算符主要有 6 个：`==`或 `eq`（等于）、`!=`或 `ne`（不等于）、`<`或 `lt`（小于）、`>`或 `gt`（大于）、`<=`或 `le`（小于等于）、`>=`或 `ge`（大于等于），如表 10-2 所示。

表 10-2 关系运算符

关系运算符	说 明	示 例	示例结果
<code>==</code> 或 <code>eq</code>	等于	<code>{5 == 5}</code> 或 <code>{5 eq 5}</code>	true
<code>!=</code> 或 <code>ne</code>	不等于	<code>{5 != 5}</code> 或 <code>{5 ne 5}</code>	false
<code>&lt;</code> 或 <code>lt</code>	小于	<code>{3 &lt; 5}</code> 或 <code>{3 lt 5}</code>	true
<code>&gt;</code> 或 <code>gt</code>	大于	<code>{3 &gt; 5}</code> 或 <code>{3 gt 5}</code>	false
<code>&lt;=</code> 或 <code>le</code>	小于等于	<code>{3 &lt;= 5}</code> 或 <code>{3 le 5}</code>	true
<code>&gt;=</code> 或 <code>ge</code>	大于等于	<code>{3 &gt;= 5}</code> 或 <code>{3 ge 5}</code>	false

书写带有关系运算符的 EL 表达式时，要注意书写的格式，如判断输入的两个密码是否相等，不能写成：

```
{param.password1} = = {param.password2}
```

而应写成：

```
{param.password1} = = param.password2}
```

关系运算符不仅可以用于数值型数据之间的比较，也可用于字符串数据的比较，如果比较的两个数据其中一个为 `String` 类型，则另一个也必须转换为 `String` 后再做比较。

### 4. 逻辑运算符

EL 的逻辑运算符有 3 个：`&&`或 `and`（而且）、`||`或 `or`（或）、`!`或 `not`（非），如表 10-3 所示

表 10-3 逻辑运算符

逻辑运算符	说 明	示 例	示例结果
假设 A 的初始值为 true，B 的初始值为 false			
<code>&amp;&amp;</code> 或 <code>and</code>	而且	<code>{A &amp;&amp; B}</code> 或 <code>{A and B}</code>	false
<code>  </code> 或 <code>or</code>	或	<code>{A    B}</code> 或 <code>{A or B}</code>	true
<code>!</code> 或 <code>not</code>	非	<code>{!A}</code> 或 <code>{not A}</code>	false

### 5. 其他运算符

`empty` 运算符用来判断值是否为 `null` 或空，如：

```
{empty param.name}
```

要判断是否为空的变量可以是字符串、数组、`Map` 及 `Collection` 等数据类型，如果需要其不为空，则在 `empty` 前加 `not`，即：

```
{not empty param.name}
```



条件运算符 “?:” 为三目运算符，即参与运算的运算数有 3 个，如下所示：


```
${A?B:C}
```

如果 A 为 true 则回传 B 的结果，否则回传 C 的结果。

“()” 括号运算符用来改变 EL 表达式中的优先级。

## 6. 运算符的优先级

[]  
 ()  
 - (负)、not、!、empty  
 \*, /、div、%、mod  
 +、- (减)  
 <、>、<=、>=、lt、gt、le、ge  
 ==、!=、eq、ne  
 &&、and  
 ||、or  
 ?:



### 10.1.2 保留字

EL 的保留字有如下一些：

and、or、no、instanceof、eq、ne、lt、empty、gt、le、ge、div、true、false、null、mod  
 当在给 EL 变量命名时，应避免使用这些保留字。

### 10.1.3 变量查找范围

EL 表达式中可以指定变量的有效范围以方便快速定位和存取变量，如果没有指定有效范围，则默认会先从 page 范围查找，如果找不到再依次从 request、session、applicaton 查找，即：page→request→session→applicaton，如果在此过程中找到了变量将不再找下去。

### 10.1.4 自动类型转换

在 EL 表达式中，变量会自动做数据类型的转换，来看如下的 EL 表达式：

```
${param.pageCount+1}
```

假设 pageCount 是从一个页面提交来的参数，页面之间传递的数据都是字符串类型的数据，上面的表达式中自动做了一个从字符串到数字数据的转换，并完成加法运算。这个 EL 表达式相当于 Java 语句中的三句：

```
String pageCountStr=request.getParameter("pageCount");
int pageCount=Integer.parseInt(pageCountStr);
pageCount++;
```

EL 表达式中的变量类型转换遵循如下规则。

(1) 变量 A 要转换为 String 类型时的规则: 如果 A 类型为 String, 直接回传 A; 如果 A 为 null, 则回传 “”, 即长度为 0 的字符串; 否则, 回传 A.toString(), 如果出现异常则报错, 没有异常则回传 A.toString()。

(2) 变量 A 要转换为数字型数据到数字型变量 N 时的规则: 如果 A 值为 null 或 “”, 则回传 0; 如果 A 为字符型数据, 将 A 作 new Short((Short)a.charValue()) 转换; 如果 A 为布尔型数据, 直接报错; 如果 A 为数字型数据且和 N 数据类型一致时, 回传 A; 如果 A 为数字型数据但和 N 数据类型不同时:

a. 如果 N 为 BigInteger 时: 如果 A 为 BigDecimal, 回传 A.toBigInteger(); 否则, 回传 BigInteger.valueOf(A.longValue())。

b. 如果 N 为 BigDecimal 时: 如果 A 为 BigInteger, 回传 A.toBigDecimal(); 否则, 回传 BigDecimal.valueOf(A.doubleValue())。

c. 如果 N 为 Byte, 回传 new Byte(A.byteValue())。

d. 如果 N 为 Short, 回传 new Short(A.shortValue())。

e. 如果 N 为 Integer, 回传 new Integer(A.intValue())。

f. 如果 N 为 Long, 回传 new Long(A.longValue())。

g. 如果 N 为 Float, 回传 new Float(A.floatValue())。

h. 如果 N 为 Double, 回传 new Double(A.doubleValue())。

i. 以上条件均不满足时, 报错。

如果 A 为 String 时:

a. 如果 N 为 BigDecimal 时: 如果 new BigDecimal(A) 发生异常, 报错; 否则回传 new BigDecimal(A)。

b. 如果 N 为 BigInteger 时: 如果 new BigInteger(A) 发生异常, 报错; 否则回传 new BigInteger(A)。

c. 如果 N.valueOf(A) 发生异常, 报错; 否则, 回传 N.valueOf(A)。

如果以上两种中的情况均不满足, 报错。

(3) 变量 A 要转换为 Character 类型数据时的规则: 如果 A 为 null 或 “”, 回传(char)0; 如果 A 为 character 类型, 直接回传 A; 如果 A 为 Boolean 时, 报错; 如果 A 为 Number 时, 转换为 Short 后, 再回传其 character; 如果 A 为 String, 回传 A.charAt(0); 以上情况都不是, 则报错。

(4) 变量 A 要转换为 Boolean 类型数据时的规则: 如果 A 为 null 或 “”, 回传 false; 如果 A 为 Boolean 时, 回传 A; 如果 A 为 String, 且 Boolean.valueOf(A) 转换正常时, 回传 Boolean.valueOf(A); 其他情况均报错。

### 【实例 10-1】使用 EL 表达式

useEL.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
```





续表

隐含对象名称	对应的类	说 明
param	java.util.Map	同 public String ServletRequest.getParameter (String name), 与输入有关
paramValues	java.util.Map	同 public String[] ServletRequest.getParameter (String name), 与输入有关
header	java.util.Map	同 public String ServletRequest.getHeader (String name)
headerValues	java.util.Map	同 public String[] ServletRequest.getHeaders (String name)
cookie	java.util.Map	同 public Cookie[] HttpservletRequest. getCookies()
initParam	java.util.Map	同 public String ServletContext. getInitParameter(String name)

通过隐含对象来得到某个变量的值，用法如下：

```
${requestScope.username}
```

此 EL 表达式得到 request 范围内的 username 变量的值。

在 EL 表达式的 11 个隐含对象中，pageScope、requestScope、sessionScope、applicationScope 这 4 个对象与变量的有效范围有关，可以用来得到相应范围内变量的值。如 session 范围内有一个变量 username，用 Java 的表达式为：

```
session.getAttribute("username")
```

改用 EL 表达式则为：

```
${sessionScope.username}
```

在 EL 表达式的 11 个隐含对象中，与输入有关的有两个对象：param 和 paramValues。以前，得到提交的表单中的数据使用如下语句：

```
request.getParameter(String name)
request.getParameterValues(String name)
```

改用 EL 表达式可用如下语句：

```
${param.name}
${paramValues.name}
```

如果要得到某个 cookie 的值，可以使用如下的形式：

```
${cookie.cookieName}
```

header 和 headerValues 这两个对象用来得到客户端与服务端交互的头文件中的一些数据，使用如下的形式：

```
${header["头文件中的数据项名"]}
```



**提示** 使用 “[ ]” 运算符可有效避免出错的可能，因为有的数据名称可能含有字符 “-”，如 User-Agent。

initParam 这个对象可用来得到当前 Web 应用在 web.xml 文件中配置的一些环境参数的值。假设当前 Web 应用的 web.xml 文件中有如下一段：

```
<context-param>
  <param-name>username</param-name>
  <param-value>myname</param-value >
</context-param>
```



得到此参数的 Java 语句为:

```
String username=(String)application.getInitParameter("username")
```

改用 EL 表达式语句为:

```
${initParam.username}
```

通过 `pageContext` 对象可以得到有关用户请求及当前页面的详细信息, 比较常用的有:

- a. `${pageContext.request.queryString}`: 得到请求字符串。
- b. `${pageContext.request.requestURL}`: 得到请求的 URL, 不包括其中的参数字符串。
- c. `${pageContext.request.contextPath}`: 得到当前 Web 应用的名称。
- d. `${pageContext.request.method}`: 得到 HTTP 的方法 (GET、POST 等)。
- e. `${pageContext.request.protocol}`: 得到使用的协议 (HTTP/1.1、HTTP/1.0)。
- f. `${pageContext.request.remoteUser}`: 得到用户名。
- g. `${pageContext.request.remoteAddr}`: 得到用户的 IP 地址。
- h. `${pageContext.session.new}`: 是否是一个新的会话。
- i. `${pageContext.session.id}`: 得到当前会话的 ID。
- j. `${pageContext.servletContext.serverInfo}`: 得到主机的相关信息。

### 【实例 10-2】使用 EL 表达式获得表单中的数据

在此例中, 一起来设计一个输入表单页面 `requestForm.jsp`, 用于输入用户名和密码, 提交表单后, 数据提交到接收数据的页面 `acceptForm.jsp`, 在 `acceptForm.jsp` 显示接收到的表单中的数据。

`requestForm.jsp`

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
<title>请求表单</title>
</head>
<body>
<form action="acceptForm.jsp" method="post">
<table border=1>
<tr><td colspan=2 align=center>输入数据的表单</td></tr>
<tr>
<td>用户名: </td>
<td><input type="text" name="username"></td>
</tr>
<tr>
<td>密码: </td>
<td><input type="password" name="password"></td>
</tr>
<tr><td colspan=2 align=center>
<input type="submit" value="提交">
</td></tr>
</table>
</form>
</body>
</html>
```

程序运行结果如图 10-2 所示。



图 10-2 输入数据的表单

这个页面中的代码定义了一个表单，表单中主要有两个输入项，一个输入用户名，一个输入密码，这在用户登录的操作中比较常见。

#### acceptForm.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head>
    <title>接收表单数据</title>
</head>
<body>
    接收到的用户名: ${param.username}<br>
    接收到的密码: ${param.password}
</body>
</html>
```

在图 10-2 中输入数据后，单击“提交”按钮，页面跳转到 `acceptForm.jsp`，如图 10-3 所示。

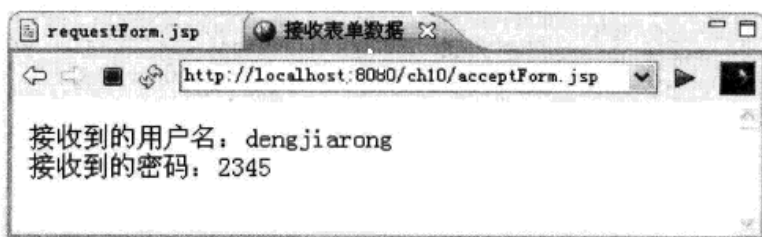


图 10-3 接收数据的页面

在 `acceptForm.jsp` 中，接收数据的 EL 表达式使用了隐含对象 `param`。

## 10.3 用 EL 操作 JavaBean

### 【实例 10-3】通过 EL 表达式使用 JavaBean

下面开发一个简单的 JavaBean——Box，并用 EL 来使用。Box 类的代码如下所示。



## Box.java

```

package javabean;
public class Box {
    long length=01;
    long width=01;
    long height=01;
    public long getLength() {
        return length;
    }
    public void setLength(long length) {
        this.length = length;
    }
    public long getWidth() {
        return width;
    }
    public void setWidth(long width) {
        this.width = width;
    }
    public long getHeight() {
        return height;
    }
    public void setHeight(long height) {
        this.height = height;
    }
    /*求表面积*/
    public long getArea(){
        return (length*width+width*height+height*length)*2;
    }
    /*求容积*/
    public long getVolumn(){
        return length*width*height;
    }
}

```

## useBox.jsp

```

<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<title>用 EL 使用 JavaBean</title>
</head>
<body>
<%//-----生成一个 JavaBean-----
    javabean.Box box=new javabean.Box();
    box.setLength(10);
    box.setWidth(20);
    box.setHeight(30);
    request.setAttribute("box",box);
%>
盒子的表面积为: ${box.area}<br>
<c:set var="volumn" value="volumn"/>
盒子的容积为: ${box[volumn]}
</body>
</html>

```

程序运行结果如图 10-4 所示。

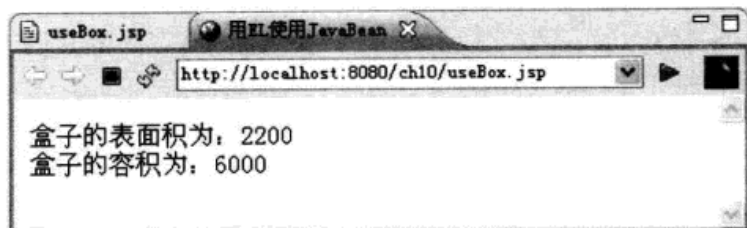


图 10-4 用 EL 使用 JavaBean

程序在用 Java 语句生成 JavaBean 对象 box 后, 再用 `setXxxx()` 方法设置了属性的初始值, 将对象放入 request 范围中, 此后 EL 表达式即可使用 box 对象了。

为了得到盒子的表面积, 使用了如下的方法:

```
${box.area}
```

读者看起来可能觉得很奇怪, box 对象并没有 area 对象啊? 是的, 没有关系, 因为表达式实际上调用的是 `getArea()` 方法, 从而得到盒子的表面积。如果使用下面的表达式, 将是错误的:

```
${box.getArea()}
```

此表达式将会抛出如下的异常:

```
The function getArea must be used with a prefix when a default namespace is not specified
```

## 10.4 小结

有了 EL, 在 JSP 页面中可用它与 JSTL 或自定义标签, 可以构建出几乎没有 Java 程序片的页面, 代码量大为减少, 可读性较强, 这为程序员及维护者带来方便。

EL 的语法简单, 应用方便, 其语法结构也比较容易理解。EL 表达式均以 “`${`” 开始, 以 “`}`” 结束。

应用 EL 的 11 个隐含对象, 可得到某个变量的值, 其中, `pageScope`、`requestScope`、`sessionScope`、`applicationScope` 这 4 个对象与变量的有效范围有关, 可以用来得到相应范围内变量的值; 与输入有关的有两个对象: `param` 和 `paramValues`。

## 10.5 练习

1. 编写一个用户注册表单, 至少包括“用户名、密码、再输入一次密码、性别、出生年月及住址”输入项, 在接收数据的 JSP 页面中用 EL 表达式得到并显示输入项的值。

2. 编写一个 JavaBean, 生成其属性的 `setXxxx()` 方法和 `getXxxx()` 方法, 结合使用 JSTL 和 EL 在 JSP 页面中生成和使用它。(程序代码可参考实例 10-3)。



# 11

## 常用开发功能实现

本章中将和读者一起分享一些 Java Web 开发中常用的实用功能，包括 XML 文件的操作，文件的上传和下载、Web 报表与图形的制作、Email 的发送和接收等，读者掌握这些技巧，将给 Java Web 系统的开发带来实质性的帮助。

### 11.1 操作 XML 文件

XML 是一套定义语义标记的规则，这些标记将文档分成许多部件并对这些部件加以标识。它也是元标记语言，即定义了用于定义其他与特定领域有关的、语义的、结构化的标记语言的句法语言。

#### 11.1.1 XML 概述

XML 文件在实践工程中主要应用于三个方面。

(1) 作为数据的表述。用 XML 文件可以直接存储数据，就像是一个数据库，对 XML 文件可以进行各种类似于数据库的操作，比如插入、删除、修改。不过 XML 文件并不能像数据库那样有丰富的功能，比如存储过程、事务处理等。

(2) 作为系统的配置文件。由于 XML 有良好的数据结构，许多 Java 领域的软件、构件使用 XML 文件来作为配置文件，以保持系统的柔性。

(3) 作为数据交换的中间文件。由于传统的数据交换报文在数据结构的表現上，数据约束的功能相对较弱，而且跨平台时要考虑诸多的问题，比如字符编码的问题、报文格式的问题等，使用 XML 文件将可以很好的解决上述问题。

鉴于以上三个方面的应用，XML 正被广泛地应用。有关 XML 的学术研究也正在兴起。如 XML 数据库的研究，XML 文件压缩的研究等。

XML 只描述结构和语义，并不说明数据在浏览器中显示的格式。如果要把 XML 的内容展示在浏览器中，可结合 XSL (eXtensible Stylesheet Language, 可扩展样式表语言) 来实现。如

果觉得 XML 定义的数据元素值不能过于随意,可用 DTD 或 XML Schema 来做约束,推荐使用 XML Schema,因为它完全使用 XML 方式表达,相对 DTD 具有良好的可读性,而且支持许多常用的数据类型。



**提示** 严格地说来,XML 文档与 XML 文件是不同的概念,XML 文档指的是 XML 形式的数据在内存中的数据表现,而 XML 文件是 XML 文档在操作系统中的物理表现,比如一个 XML 文件被加载后就成了 XML 文档。在本章中不严格地区分这些概念,认为指的就是 XML 表现的数据形式。

### 11.1.2 XML 文件的结构

其实在本书前面的章节中已经接触过 XML 了,在 Tomcat 的配置文件中,如: server.xml、web.xml 都是使用 XML。为了便于理解,特别是初学者,下面一起来看一个简单的 XML 文件,文件内容如下(编辑工具可以用记事本、写字板等任一支持文本编辑的工具):

user.xml

```
<?xml version="1.0" encoding="gb2312"?>
<users>
  <user>
    <id>1</id>
    <name>dzy</name>
    <password>6582254</password>
    <true_name>邓子云</true_name>
    <age>26</age>
    <sex>男</sex>
    <address>长沙市商业银行信息技术部</address>
    <telephone>0731-4305524</telephone>
    <add_time>2006-4-2 15:20:23</add_time>
  </user>
</users>
```

文件内容的第一行:

```
<?xml version="1.0" encoding="gb2312"?>
```

这是 XML 处理指令声明的第一句,以“<?”开始,以“?”结束,每个 XML 文件都要以一句声明开始。本句中带有 version 和 encoding 两个特性声明,version 表示版本号,encoding 表示解码方式,这里用“gb2312”,以方便显示中文。

此句之后的标记又是什么含义呢?那就要看编写文档的人了。建议标记名称用容易理解的名称,如本文中<users></users>表示所有的用户。编写标记时需要配对使用,如“<users>”与“</user>”配对、“<id>”与“</id>”配对。

如果要换一种表达方式,可把下一级结点的内容改为在属性中表示。如把 user.xml 稍作改写,内容如下:

```
<?xml version="1.0" encoding="gb2312"?>
<users>
  <user id="1" name="dzy" password="6582254" true_name="邓子云" age="26"
```



```
sex="男" address="长沙市商业银行信息技术部" telephone="0731-4305524"
add_time="2006-4-2 15:20:23">
</user>
</users>
```

与前面的 user.xml 表达的数据内容是一样的。如果标记之处没有内容，标记的配对也可以简写，书写的方法就是在开始标记后加“/”就可以了。如上面的第二句可改写为：

```
<user id="1" name="dzy" password="6582254" true_name="邓子云" age="26"
sex="男" address="长沙市商业银行信息技术部" telephone="0731-4305524"
add_time="2006-4-2 15:20:23"/>
```

### 11.1.3 DTD 文档

DTD 文档用来对 XML 文件定义一些约束，比如 XML 文件中的元素、元素的属性、元素的排列方式和顺序、元素能够包含的内容等。XML 文件中的元素，即我们所创建的标记，是根据应用的实际情况来创建的。想要创建一份完整性高、适应性广的 DTD 是非常困难的，因为各行各业都有他们自己的行业特点，所以 DTD 通常是以某种应用领域为定义的范围，如：医学、建筑、工商、行政。

DTD 可以是一个完全独立的文件，也可以在 XML 文件中直接设定。所以，DTD 分为外部 DTD（在 XML 文件中调用另外已经编辑好的 DTD）和内部 DTD（在 XML 文件中直接设定 DTD）两种。比如，有几十家相互联系的、合作伙伴关系的公司、厂商，他们相互之间的交换电子文档都是用 XML 文档。那么我们可以将这些 XML 文档的 DTD 放在某个地方，让所有交换的 XML 文档都使用此 DTD，这是最方便的做法，同时也适用于公司内部 XML 文件使用。

#### 1. 内部 DTD

内部 DTD 是在 XML 文件的文件首部中定义的，使用的语法如下：

```
<!DOCTYPE 根元素名[.....]>
```

<!DOCTYPE：表示开始设定 DTD，注意 DOCTYPE 是大写。

根元素名：一个 XML 文件只能有一个根元素。注意，如果 XML 文件使用了 DTD，那么文件中的根元素就在这里指定。

[.....]>：在[]标记里面定义 XML 文件使用元素，然后用>结束 DTD 的定义。

DTD 内容定义的主要语法如下：

```
<!ELEMENT 元素名 元素定义>
```

<!ELEMENT：表示开始元素设置，注意此处 ELEMENT 关键字是大写。

元素定义就是指<元素>...</元素>之间能够包含什么内容，是其他元素还是一般性的文字。

#### 2. 外部 DTD

声明外部 DTD 需要编写 DTD 文件，扩展名为 dtd，在 XML 文档 DOCTYPE 声明中添加 DTD 引用，告诉解析器外部 DTD 信息，语法如下：

```
<!DOCTYPE 根元素名 SYSTEM "DTD 文件名.dtd">
```

### 3. 元素的定义

元素的定义有如下几种形式。

#### (1) 带有数据的元素。

```
<! ELEMENT 元素名 (数据类型)>
```

数据类型：

**#CDATA:** 指元素包含不通过解析器解析的字符数据。特殊字符和保留字不需要转义。

**#PCDATA:** 指元素包括解析器可解析字符数据。特殊字符和保留字需要转义才可以通过解析器。

**ANY:** 元素可以包含任何声明类型的子元素和字符数据。

#### (2) 带有子元素的元素。

```
<! ELEMENT 元素名 (子元素 1 名,子元素 2 名)>
```

多个子元素用逗号隔开。在文档中的顺序和定义中的顺序一致。子元素可以有自己的子元素。如果相同元素只出现一次，使用如下的定义形式：

```
<! ELEMENT 元素名 (子元素)>
```

如果相同元素至少出现一次，使用如下的形式：

```
<! ELEMENT 元素名 (子元素+)>
```

如果相同元素出现零次或多次，使用如下的形式：

```
<! ELEMENT 元素名 (子元素*)>
```

如果相同元素出现零次或一次，使用如下的形式：

```
<! ELEMENT 元素名 (子元素?)>
```

#### (3) 空元素

```
<! ELEMENT 元素名 (EMPTY)>
```

空元素虽然没有子元素，但是可以有属性。

#### (4) 混合声明

组可以是序列、选择子元素和子组。序列的声明形式有：

**<! ELEMENT A (B)>**：元素 A 由单个子元素 B 组成；

**<! ELEMENT A (B, C)>**：元素 A 由子元素 B 和 C 组成；

**<! ELEMENT A (B, (C | D), E)>**：元素 A 由子元素 B, E 和选择子组(C 或 D 中之一)组成。

选择子元素和子组的声明形式有：

**<! ELEMENT A (B | C)>**：元素 A 由子元素选择子组(B 或 C)组成；

**<! ELEMENT A (B | C | (D, E))>**：元素 A 由 B、C 或序列(D,E)组成。

### 4. 属性的声明

#### (1) 空属性

```
<! ATTLIST 元素名 EMPTY>
```

#### (2) 非空属性

```
<! ATTLIST 元素名 属性名 属性类型 属性值>
```



属性值有以下的定义形式：

① **Default** 属性值：指定一个默认值。

```
<! ATTLIST 元素名 属性名 属性类型 "默认值" >
```

DTD 示例：

```
<! ATTLIST 售价 货币单位 CDATA "人民币">
```

XML 示例：

```
<售价 货币单位 = "人民币">10.90</售价>
```

② **Implied** 属性值：可以不提供该属性，该属性也没有默认值。

```
<! ATTLIST 元素名 属性名 属性类型 #IMPLIED >
```

DTD 示例：

```
<! ATTLIST 售价 货币单位 CDATA #IMPLIED>
```

XML 示例：

```
<售价>10.90</售价>
```

③ **Required** 属性值：必须提供该属性，但可以没有默认值。

```
<! ATTLIST 元素名 属性名 属性类型 #REQUIRED >
```

DTD 示例：

```
<! ATTLIST 售价 货币单位 CDATA #REQUIRED>
```

XML 示例：

```
<售价 货币单位 = "美元">10.90</售价>
```

④ **Fixed** 属性值：使属性具有固定值，不可以更改。

```
<! ATTLIST 元素名 属性名 属性类型 #FIXED "固定值">
```

DTD 示例：

```
<! ATTLIST 售价 货币单位 CDATA #FIXED "人民币">
```

XML 示例：

```
<售价 货币单位 = "人民币">10.90</售价>
```

(3) 属性类型

① **Enumerated** 属性类型：使默认值成为一组固定值中之一。

```
<! ATTLIST 元素名 属性名 (固定值 A|固定值 B|...) 默认固定值之一>
```

DTD 示例：

```
<! ATTLIST 售价 货币单位 ("人民币"|"美元"|"欧元") "人民币">
```

XML 示例：

```
<售价 货币单位 = "欧元">10.90</售价>
```

② **ID** 和 **IDREF** 属性类型

ID 用于搜索某个元素的特定实例，每个元素都可以具有 ID 类型的属性。

```
<! ATTLIST 元素名 属性名 ID 属性值>
```

DTD 示例:

```
<! ATTLIST 售价 支付方式 ID #REQUIRED>
```

XML 示例:

```
<售价 ID = "BOOK1">10.90</售价>
<售价 ID = "BOOK2">10.90</售价>
<售价 ID = "BOOK3">10.90</售价>
```

IDREF 用于指向一个元素, 引用其他元素中的一个元素。

```
<! ATTLIST 元素名 属性名 IDREF 属性值>
```

DTD 示例:

```
<! ATTLIST 售价 货币单位 ID #REQUIRED>
<! ATTLIST 售价 货币单位 IDREF #IMPLIED>
<! ATTLIST 售价 货币单位 CDATA #IMPLIED>
```

XML 示例:

```
<售价 ID = "BOOK1" 货币单位 = "人民币">10.90</售价>
<售价 ID = "BOOK2" IDREF = "BOOK1"></售价>
```

### ③ IDREFS 属性类型

指向多个元素 ID, 用空格分开。用于指向 XML 文档中的相关元素列表。

```
<! ATTLIST 元素名 属性名 IDREFS 属性值>
```

## 【实例 11-1】user.xml 文件的 DTD 文档

我们仍然使用 11.1.1 节中介绍的 user.xml 文件, 下面为其编写 DTD 文档。如果使用外部 DTD, 则在 user.xml 文件中应增加声明语句, 如下所示。

user.xml

```
<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE users SYSTEM "user.dtd">
<users>
  <user>
    <id>1</id>
    <name>dzy</name>
    <password>6582254</password>
    <true_name>邓子云</true_name>
    <age ageunit="岁">26</age>
    <sex>男</sex>
    <address>长沙市商业银行信息技术部</address>
    <telephone>0731-4305524</telephone>
    <add_time>2006-4-2 15:20:23</add_time>
  </user>
</users>
```

这里将 DTD 文档指向了 user.dtd。来看 user.dtd 文档的内容。

user.dtd

```
<?xml version="1.0" encoding="gb2312"?>
<!ELEMENT users (user+)>
```



```

<!ELEMENT user (id,name,password,true_name,age,sex,address,telephone,add_time)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT true_name (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT sex (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>
<!ELEMENT add_time (#PCDATA)>
<!ATTLIST age ageunit CDATA "岁">

```

从 DTD 文档来看, 声明了各个元素的构成, 以及 age 元素的属性 ageunit, 默认单位为“岁”。user.xml 文件在浏览器中运行的结果如图 11-1 所示。



图 11-1 user.xml 文件在浏览器中运行的结果

如果要将 DTD 文档作为内部 DTD, 则应在 user.xml 文件的首部加如下的内容:

```

<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE users[
<!ELEMENT users (user+)>
<!ELEMENT user (id,name,password,true_name,age,sex,address,telephone,add_time)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT true_name (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT sex (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>
<!ELEMENT add_time (#PCDATA)>
<!ATTLIST age ageunit CDATA "岁">
]>
<users>
.....
</users>

```

## 11.1.4 XML Schema

XML Schema 是 2001 年 5 月正式发布的 W3C 的推荐标准。XML Schema 语言也被称为 XSD (XML Schema Definition, XML Schema 定义), 文件的扩展名为 .xsd。XML Schema 相比 DTD 有很多优点, 它完全使用 XML 作为描述手段, 具有很强的描述能力、扩展能力和处理维护能力。它支持值的类型例如: 整数、串等对应数据库中的预定义的值域类型, 支持对元素出现最小次数或最大次数加以限制, 支持自定义的数据类型, 支持数据元素的继承等。

在 XML 文档中加入 XML Schema 检验需要在 XML 文件的首部使用如下的语句:

```
<users xmlns="http://www.56edu.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.56edu.com user.xsd">
```

其中 users 是根结点, xmlns="http://www.w3school.com.cn" 规定了默认命名空间的声明。此声明会告知 schema 验证器, 在此 XML 文档中使用的所有元素都被声明于 "http://www.56edu.com" 这个命名空间。

一旦您拥有了可用的 XML Schema 实例命名空间:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

您就可以使用 schemaLocation 属性了。此属性有两个值。第一个值是需要使用的命名空间。第二个值是供命名空间使用的 XML schema 的位置:

```
xsi:schemaLocation="http://www.56edu.com user.xsd"
```

XSDL (XML Schema 定义语言) 由元素、属性、命名空间和 XML 文档中的其他节点构成。

### 1. XSD 中的元素

XSD 文档至少要包含: schema 根元素和 XML 模式命名空间的定义、元素定义。

#### (1) schema 根元素

语法如下:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
... ..
</xsd:schema>
```

在 XSD 中必须定义一个且只能定义一个 schema 根元素。根元素中包括模式的约束、XML 模式命名空间的定义, 其他命名空间的定义、版本信息、语言信息和其他一些属性。

#### (2) 元素

语法如下:

```
<xsd:element name="user" type="xsd:string" />
```

XSD 中的元素是利用 element 标识符来声明的。其中 name 属性是元素的名字, type 属性是元素值的类型, 在这里可以是 XML Schema 中内置的数据类型或其他类型。下面是一个例子:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="user" type="xsd:string" />
</xsd:schema>
```

以上文档对应的有效 XML 文档如下:



```
<?xml version="1.0"?>
<user>string</user>
```

在元素的定义中还有两个属性：`minOccurs` 和 `maxOccurs`。其中 `minOccurs` 定义了该元素在父元素中出现的最少次数（默认为 1，值为大于等于 0 的整数），`maxOccurs` 定义了该元素在父元素中出现的最大次数（默认为 1，值为大于等于 0 的整数）。在 `maxOccurs` 中可以把值设置为 `unbounded`，表示对元素出现的最大次数没有限制。下面是一个例子：

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="user" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
</xsd:schema>
```

表示元素 `user` 的类型为 `string`，出现的次数最少为 0（也就是可选），最多不限制。

### (3) 引用元素和替代

语法如下：

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="user" type="xsd:string" />
  <xsd:element name="name">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="user" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

引用是利用 `element` 标记符的 `ref` 属性实现的。主要适用于避免在文档中多次定义同一个元素，应当将经常使用的元素定义为根元素的子元素，以便在文档的任何地方引用它。在这里还可以为某个定义的元素起一个别名，方法如下：

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="yonghu" type="xsd:string" substitutionGroup="user" />
  <xsd:element name="user" type="xsd:string" />
  <xsd:element name="name">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="user" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

以上文档对应的有效 XML 文档如下：

```
<?xml version="1.0"?>
<name>
  <user>string</user>
</name>
```

或者

```
<?xml version="1.0"?>
<name>
  <yonghu>string</yonghu>
</name>
```

主要是利用 **element** 标识符中的属性 **substitutionGroup** 实现的。

#### (4) 设置默认值和固定值

语法如下：

```
<xsd:element name="city" type="xsd:string" default="xian" />
<xsd:element name="country" type="xsd:string" fixed="china" />
```

通过 **default** 属性的设置，可以在 XML 文档中没有对 **city** 元素定义时赋予默认值。而使用 **fixed** 属性，可以给元素 **country** 设定一个固定的值 **china**，并且不允许改变。

#### (5) 利用组合器控制结构

**sequence** 组合器，定义了一系列元素必须按照模式中指定的顺序显示（如果是可选的，也可以不显示）。语法如下：

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="name">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="first" type="xsd:string" />
        <xsd:element name="middle" type="xsd:string" />
        <xsd:element name="last" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

**all** 组合器，允许所定义的元素可以按照任意顺序显示，**all** 元素的子元素在默认情况下是必须的，而且每次最多显示一次。语法如下：

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="name">
    <xsd:complexType>
      <xsd:all minOccurs="0">
        <xsd:element name="first" type="xsd:string" />
        <xsd:element name="middle" type="xsd:string" />
        <xsd:element name="last" type="xsd:string" />
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

**choice** 组合器，允许指定多组声明中的一个，用于互斥情况。语法如下：

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="name">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element name="first" type="xsd:string" />
        <xsd:element name="middle" type="xsd:string" />
        <xsd:element name="last" type="xsd:string" />
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



## 2. 属性

在 XML Schema 文档中可以按照定义元素的方法定义属性，但受限制的程度较高。它们只能是简单类型，只能包含文本，且没有子属性。可以应用在 **attribute** 元素定义中的属性如下：

- **default**: 初始默认值。
- **fixed**: 不能修改和覆盖的属性固定值。
- **name**: 属性的名称。
- **ref**: 对前一个属性定义的引用。
- **type**: 该属性的 XSD 类型或者简单类型。
- **use**: 如何使用属性。
- **form**: 确定 **attributeFormDefault** 的本地值。
- **id**: 模式文档中属性唯一的 ID。

**default**、**fixed**、**name**、**ref** 和 **type** 属性与在 **element** 标记中定义的对属性相同，但 **type** 只能是简单类型。**use** 属性的值可以是：**optional**（属性不是必须的，此为默认属性）、**prohibited** 或者 **required**（属性是强制的）。

创建属性语法如下：

```
<xsd:attribute name="age" type="xsd:integer" />
```

该语句定义了一个名为 **age** 的属性，它的值必须是一个整数。把它添加到模式中时，它必须是 **schema** 元素、**complexType** 元素或者 **attributeGroup** 元素的子元素。下面是一个例子：

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="name">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="first" type="xsd:string" />
      </xsd:sequence>
      <xsd:attribute name="age" type="xsd:integer" use="optional" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

以上文档对应有效的 XML 文档如下：

```
<?xml version="1.0"?>
<name age="27">
  <first>string</first>
</name>
```

如上所示，要把属性附加在元素上，属性应该在 **complexType** 定义中的组合器之后定义或引用。

## 3. XML Schema 数据类型

XML Schema 提供了一组丰富的内置数据类型，用于定义元素中允许的类型。下面将介绍一些常用的通用类型。

### (1) 基本数据类型

基本数据类型是在 XML Schema 中使用的每种数据类型的最基本构成块。可以根据这些类型构造自定义的类型。这些类型包括：

**boolean**：可以是 1 (true) 或者 0 (false)。

**dateTime**：表示时间的部分可选，格式：CCYY-MM-DDThh:mm:ss，例如：2005-3-18T14:48:12。

**decimal**：表示任意精度的十进制数字。

**string**：字符数据。

**int**：表示从 -2,147,483,648 到 2,147,483,648 之间一个整数。

**nonNegativeInteger**：表示大于或者等于 0 的一个整数。

**nonPositiveInteger**：表示小于或者等于 0 的一个整数。

**short**：表示从 -32768 到 32767 之间的一个整数。

例如：

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
  <xsd:element name="ContactDetails">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="name" />
        <xsd:element name="rate" type="xsd:decimal" />
      </xsd:sequence>
      <xsd:attribute name="lastUpdated" type="xsd:dateTime" />
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="name">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="first" type="xsd:string" />
        <xsd:element name="middle" type="xsd:string" />
        <xsd:element name="last" type="xsd:string" />
      </xsd:sequence>
      <xsd:attribute name="local" type="xsd:boolean" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

以上文档对应有效的 XML 文档如下：

```
<?xml version="1.0"?>
<ContactDetails lastUpdated="2005-3-18T14:48:12">
  <name local="true">
    <first>santld</first>
    <middle/>
    <last>wang</last>
  </name>
  <rate>10.27</rate>
</ContactDetails>
```

### (2) 约束

虽然从内置数据类型中得到了许多的功能，但是在许多情况下，只有数据类型来限制数据的值是远远不够的。这里在学习简单类型之前，先看看关于约束。



**enumeration**: 用空格分开的一组指定的数值。它把数据类型约束为指定的值。

**fractionDigit**: 指定小数点后的最大位数。

**length**: 长度单位。

**minExclusive**: 下限值, 所有的值都必须大于该值。

**maxExclusive**: 上限值, 所有的值都应该小于该值。

**minLength**: 长度单位的最小个数。

**maxLength**: 长度单位的最大数。

**minInclusive**: 最小值, 所有的值都应该大于或者等于该值。

**maxInclusive**: 最大值, 所有的值都应该小于或者等于该值。

**pattern**: 数据类型的值必须匹配的指定模式, **pattern** 值必须是一个正则表达式。

**totalDigits**: 指定小数最大位数的值。

**whiteSpace**: **preserve** (值中的空格不能改变), **replace** (所有的制表符、换行符和回车符都用空格代替), **collapse** (执行 **replace**, 删除相邻的、结尾处和开头处的空格)

要应用上述的约束, 就要利用元素 **restriction**。这个元素中有两个属性: **id** 属性是模式文档中 **restriction** 元素的唯一标识符, **base** 属性设置为一个内置的 XSD 数据类型或者现有的简单类型定义, 他是一种被限制的类型。例如:

```
<xsd:restriction base="xsd:string">
  <xsd:minLength value="4" />
  <xsd:maxLength value="10" />
</xsd:restriction>
```

如上, 将字符串最小长度限定为 4 个字符, 将最大长度限定为 10 个字符。

```
<xsd:restriction base="xsd:int">
  <xsd:minInclusive value="1" />
  <xsd:maxInclusive value="100" />
</xsd:restriction>
```

如上, 将一个整数的取值范围设置为 1 到 100 之间。

```
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="FistValue" />
  <xsd:enumeration value="SecondValue" />
  <xsd:enumeration value="ThirdValue" />
</xsd:restriction>
```

如上, 字符串只能为枚举出的三种值。

### (3) 简单类型

简单类型是对一个节点的可能值进一步限制的自定义数据类型。创建简单类型需要利用 **simpleType** 元素, 其定义如下:

```
<simpleType id="ID" name="NCName" final="#all|(list|union|restriction)" />
```

**id** 属性应唯一的标明文档内的 **simpleType** 元素, **name** 不能使用冒号字符。 **simpleType** 不能包含元素, 也不能有属性, 根据在 **simpleType** 中定义的规则, 它基本上是一个值, 或者是一个值的集合。例如:

```
<xsd:simpleType name="personsName">
```

```

<xsd:restriction base="xsd:string">
  <xsd:minLength value="4" />
  <xsd:maxLength value="8" />
</xsd:restriction>
</xsd:simpleType>
<xsd:element name="first" type="personsName" />

```

以上文档对应有效的 XML 文档如下：

```
<first>Santld</first>
```

以下就是无效的 XML 文档：

```
<first>SS</first>
```

或者：

```
<first>Santld wang</first>
```

再举个例子：

```

<xsd:simpleType name="personsTitle">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Mr." />
    <xsd:enumeration value="Mrs." />
    <xsd:enumeration value="Miss." />
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="title" type="personsTitle" />

```

上面定义的类型 `personsTitle` 是一个字符串类型，但它的值只能是 `Mr.`、`Mrs.` 或者 `Miss.` 其中的一个。

#### (4) 复杂类型

复杂类型的定义必须使用 `complexType` 元素，在这里可以包含属性和元素。在复杂类型的使用中，主要是 `complexType` 和 `simpleType` 配合使用。格式如下：

```
<xsd:complexType name="name" />
```

例如：

```

<xsd:element name="name" type="FullName" />
<xsd:complexType name="FullName">
  <xsd:sequence>
    <xsd:element name="first" type="PersonsFirstname" minOccurs="0" maxOccurs="1"
      default="John" />
    <xsd:element name="middle" type="xsd:string" minOccurs="0"
      maxOccurs="unbounded" nillable="true" />
    <xsd:element name="last" type="xsd:string" minOccurs="1" maxOccurs="1"
      default="Doe" />
  </xsd:sequence>
  <xsd:attribute name="title" type="PersonsTitle" default="Mr." />
</xsd:complexType>
</xsd:element>
<xsd:simpleType name="PersonsFirstname">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="4" />
    <xsd:maxLength value="10" />
  </xsd:restriction>
</xsd:simpleType>

```



```

    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="PersonsTitle">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Mr." />
      <xsd:enumeration value="Mrs." />
      <xsd:enumeration value="Miss." />
    </xsd:restriction>
  </xsd:simpleType>

```

如上就实现了一个复杂类型，该例子实现了一个复杂类型 **FullName**，其中包含了两个简单类型 **PersonsFirstname** 和 **PersonsTitle**。

### (5) 分组和属性

在为定义 XML 文档而创建的一些更为复杂的定义中，会有一些元素集、属性集的组合，这时我们就使用了分组的概念。分组定义中使用的是 **group** 元素。例如：

```

<xsd:group name="CityChoice">
  <xsd:choice>
    <xsd:element name="Beijing" type="xsd:string" />
    <xsd:element name="Shanghai" type="xsd:string" />
    <xsd:element name="Xian" type="xsd:string" />
  </xsd:choice>
</xsd:group>
<xsd:element name="City">
  <xsd:group ref="CityChoice" minOccurs="1" maxOccurs="1" />
</xsd:element>

```

对属性的分组，应该使用 **attributeGroup** 元素，例如：

```

<xsd:attributeGroup name="contactAttribs">
  <xsd:attribute name="city" type="xsd:string" />
  <xsd:attribute name="country" type="xsd:string" />
  <xsd:attribute name="age" type="xsd:string" />
</xsd:attributeGroup>
<xsd:element name="contact">
  <xsd:complexType>
    <xsd:attributeGroup ref="contactAttribs" />
  </xsd:complexType>
</xsd:element>

```

以上文档对应有效的 XML 文档如下：

```

<contact city="Beijing" country="China" age="25" />

```

### (6) 内容模型

内容模型可以对在 XML 文档内使用的元素、属性和类型进行限制，确定用户可以在 XML 实例的哪些等级添加自己的元素和属性。

当在 XML 中声明元素时 **any** 是默认的内容模型，该模型可以包含文本、元素和空格。如果允许添加元素的内容，且无需修改模式文件，就可以使用该模型。例如：

```

<xsd:xschema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="name">
    <xsd:complexType>
      <xsd:sequence>

```



```

        <xsd:element name="first" type="xsd:string" />
        <xsd:element name="middle" type="OtherNames" />
        <xsd:element name="last" type="xsd:string" />
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:complexType name="OtherNames">
    <xsd:sequence>
        <xsd:any namespace="##any" processContents="lax" minOccurs="0"
            maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>

```

例子中的 `xsd:any` 元素说明该类型允许添加内容。`namespace` 属性允许的值为：`##any`（元素可以来自任何命名空间）、`##other`（元素可以来自除了该元素的父元素所在的目标命名空间之外的命名空间）、`##local`（元素不受命名空间的限制）、`##targetNamespace`（元素来自父元素的目标命名空间）。

`processContents` 属性说明对这里所创建的元素进行验证时所执行的操作，取值有如下 3 种：`strict`（标明 XML 处理器必须获得和那些命名空间相关联的模式，并验证元素和属性）、`lax`（与 `strict` 相似，只是如果处理器找不到模式文档，也不会出现错误）、`skip`（不利用模式文档验证 XML 文档）。

上述模式的一个有效实例如下：

```

<?xml version="1.0"?>
<name>
    <first>santld</first>
    <middle>
        <nameInChina>San</nameInChina>
    </middle>
    <last>wang</last>
</name>

```

`empty` 元素禁止把文本或者元素作为一个声明为空的元素的子元素，如果为了保证该元素不包括子元素、文本甚至空格，就可以使用它。在 XSD 中是利用 `xsd:anyType` 类型来限定它，这样就意味元素只能包含属性，例如：

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="contact">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:restriction base="xsd:anyType">
                    <xsd:attribute name="age" type="xsd:integer" />
                </xsd:restriction>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>

```

以上就是一个复杂类型，只允许有一个 `age` 属性。在例子里 `complexContent` 元素可以表示 `complexType` 的内容要进行扩充或者限制，在这里，我们对其内容进行限制，因此使用了



restriction 元素, 如果扩展则使用 extension 元素。

最后一个内容模型就是 mixed, 它包含文本、内容和属性。在 complexType 元素上把 mixed 属性的值设为 true, 就声明了一个 mixed 内容模型。例如:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="contact">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element name="first" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

上述模式的一个有效实例如下:

```
<?xml version="1.0"?>
<contact>
  My first name is<first>Santld</first>.
</contact>
```

在例子中 contact 元素就包含了文本和元素 first。

### 11.1.5 JDOM

解析 XML 文件的 Java 接口技术有许多, 比较常用的有 SAX、DOM、JDOM。SAX 是 Simple API for XML 的缩写, 这并非官方标准, 而是由网上社区讨论而产生的标准, 但几乎所有的 XML 解析器都会支持。用 SAX 解析 XML 文件, 无须将文档读入内存。由于 SAX 基于事件触发来解析 XML 文件, 它在解析的过程中, 不断触发事件, 因此使用者所需要做的工作就是在相应的事件方法中加入程序代码。

用 DOM 来解析 XML 文件的操作比较简便, 它的解析过程是, 首先将 XML 文件读入内存, 建立起一棵文档树, 通过对这棵文档树的操作来完成对 XML 文件的操作。因此, 如果 XML 特别大, 将占用较多的内存, 且操作数据时要做大量的遍历树的操作, 这对于大量数据的存取、修改操作率不高, 但对于数据量不大的一般性应用场合, 用 DOM 来处理已经足够了。

JDOM 是基于 Java 技术的开放源码项目, 它遵循 80/20 规则, 用 DOM 和 SAX 20% 的功能来满足 80% 的用户需求。JDOM 使用 SAX 和 DOM 解析器, 因此它是作为一组相对较小的 Java 类被实现的。

JDOM 的主要特性是它极大地减少了编写程序的代码量, JDOM 应用程序的长度通常是 DOM 应用程序的三分之一, 大约是 SAX 应用程序的一半。JDOM 是基于树操作的纯 Java API, 应该说它提供的是一套用于解析、创建、处理和实现 XML 的解决方案。这些 API 比 DOM 和 SAX API 提供的方法更为直观, 对于有 Java 经验的程序员将会发现 JDOM 非常容易掌握。JDOM 处理 XML 的方式比 DOM 容易得多, 并且它的功能比使用 SAX 更加强大。

JDOM 的内部逻辑结构基本上与 DOM 的相同, 比如具有 Document、Element、Comment 等文档结点类型, 其中每一个 JDOM 文档必须有一个 Document 结点, 并且为结点树的根结点。该根结点可以有子节点或者叶子结点, 如 Comment、Text 等。JDOM 文档中的每一个结点类型

均对应格式良好的 XML 文档中的每一个元素。这也就为利用 JDOM 转换数据库到 XML 文档提供了可操作的依据。

JDOM 具有如下特点：

(1) JDOM 是 Java 平台专用的。只要有可能，API 都使用 Java 语言的内建 String 支持，因此文本值也适用于 String。另外，它还可利用 Java 2 平台的类集，如 List 和 Iterator，给程序员提供了一个丰富的并且和 Java 语言类似的环境。

(2) 没有层次性。在 JDOM 中，XML 元素就是 Element 的实例，XML 属性就是 Attribute 的实例，XML 文档本身就是 Document 的实例。由于在 XML 中所有这些都代表了不同的概念，因此它们总是作为自己的类型被引用，而不是作为一个含糊的“结点”。

(3) 类驱动。因为 JDOM 对象就是像 Document、Element 和 Attribute 这些类的直接实例，因此创建一个新 JDOM 对象就如在 Java 语言中使用 new 操作符一样容易。它还意味着不需要进行工厂化接口配置，可直接使用 JDOM。

(4) JDOM API 比 DOM 所提供的方法更为直观，同时简化了与 XML 的交互。比使用 DOM 更快。

(5) JDOM 组合了 DOM 和 SAX 的优点。它被设计成一个可以在小内存上快速执行的轻量级 API。JDOM 也支持随机读取整个文档，但是令人惊奇的是它并不需要把整个文档读到内存中。这个 API 支持当需要时才读入信息的次轻量级操作。还有，JDOM 通过标准的构造器和 set 方法支持 XML 文档的修改。

1. JDOM 相关的类

JDOM 的包结构如表 11-1 所示。

表 11-1 JDOM 的包结构

包	说 明
org.jdom	包含了所有的 xml 文档要素的 Java 类
org.jdom.adapters	包含了与 dom 适配的 Java 类
org.jdom.filter	包含了 xml 文档的过滤器类，可以基于类型、名称、值等过滤
org.jdom.input	包含了读取 xml 文档的类
org.jdom.output	包含了写入 xml 文档的类
org.jdom.transform	包含了将 jdom xml 文档接口转换为其他 xml 文档接口
org.jdom.xpath	包含了对 xml 文档 xpath 操作的类

在 org.jdom 中，提供了 Attribute、Document、DocType、Element、Comment、Entity、Attribute、ProcessingInstruction 等 Java 类，这些类均是访问和操作 JDOM 文档所必需的。可以利用这些类创建、遍历和修改 JDOM 文档。

在 org.jdom.output 中，提供了 SAXOutputter、XMLOutputter，用于处理 JDOM 树形式、XML 文档形式输出、打印等。下面重点介绍几个常用的类及其方法。有兴趣的读者如果想要更加详细的 API 资料可参考 JDOM 的 API 文档。

在 JDOM 中，XML 元素就是 Element 的实例，XML 属性就是 Attribute 的实例，XML 文档本



身就是 Document 的实例。因为 JDOM 对象就是像 Document、Element 和 Attribute 这些类的直接实例，因此创建一个新 JDOM 对象就如在 Java 语言中使用 new 操作符一样容易。

## 2. 安装与配置 JDOM

可从如下的地址下载到 JDOM 组件：

<http://www.jdom.org/dist/binary/>

至本书成稿之日止，JDOM 的版本为 1.1。下载得到 ZIP 文件后，将其解压缩，在 build 目录下可得到 jdom.jar 组件包。

安装的过程并不复杂：直接复制 jdom.jar 文件到当前 Web 应用的“WEB-INF\lib”目录下，则对于当前 Web 应用可用；也可复制到 Tomcat 7 安装目录的 lib 目录，则所有的 Web 应用都可用。

## 3. XML 文档创建相关的操作

下面仍以本章中的 user.xml 文档为例，来看如何创建 XML 文档。

### (1) 以 users 为根元素创建文档

```
//所有的 XML 元素都是 Element 的实例，根元素也不例外。
Element rootElement = new Element("users");
//以根元素作为参数创建 Document 对象。一个 Document 只有一个根，即 root 元素。
Document myDocument = new Document(rootElement);
```

### (2) 添加元素和子元素

JDOM 里子元素是作为 content（内容）添加到父元素里面去的。

```
Element userElement = new Element("user");//创建 user 元素
rootElement.addContent(userElement);//将 user 元素作为 content 添加到根元素
Element idElement = new Element("id");//创建 id 元素
idElement.addContent("1");//将 1 作为 content 添加到 idElement
//将 idElement 元素作为 content 添加到 userElement 元素
userElement.addContent(idElement);
```

### (3) 删除子元素

这个操作比较简单：

```
rootElement.removeChild("sex");//该方法返回一个布尔值
```

### (4) 给元素添加属性

```
//给 ageElement 元素创建名为 ageunit 的属性，值为“岁”
ageElement.setAttribute(new Attribute("ageunit", "岁"));
```

或：

```
Attribute ageunitAttri=new Attribute("ageunit", "岁");
ageElement.setAttribute(ageunitAttri);
```

### (5) 将 JDOM 转化为 XML 文本

```
Format format = Format.getPrettyFormat();
format.setEncoding("gb2312");//设置解码方式
XMLOutputter xmlOut = new XMLOutputter(format);
try {
```

```

        xmlOut.output(myDocument, System.out);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

#### (6) 将 JDOM 文档转化为其他形式

XMLOutputter 还可输出到 Writer 或 OutputStream。为了输出 JDOM 文档到一个 XML 文件, 可以这样做:

```

//输出到 XML 文件
FileWriter writer = new FileWriter("d:/user3.xml");
xmlOut.output(myDocument, writer);
writer.close();

```

XMLOutputter 还可输出到字符串, 以便程序后面进行再处理:

```

String outString = xmlOut.outputString(myDocument);

```

### 【实例 11-2】使用 JDOM 创建 user.xml

#### parseUser.jsp

```

<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ page import="org.jdom.*,
                org.jdom.output.*,
                java.io.IOException,
                java.io.FileWriter" %>

<html>
<body>
<%
//所有的 XML 元素都是 Element 的实例, 根元素也不例外。
Element rootElement = new Element("users");
//以根元素作为参数创建 Document 对象。一个 Document 只有一个根, 即 root 元素。
Document myDocument = new Document(rootElement);
Element userElement = new Element("user");//创建 user 元素
rootElement.addContent(userElement);//将 user 元素作为 content 添加到根元素
Element idElement = new Element("id");//创建 id 元素
idElement.addContent("1");//将 1 作为 content 添加到 idElement
//将 idElement 元素作为 content 添加到 userElement 元素
userElement.addContent(idElement);
//====其它元素的操作====
Element nameElement = new Element("name");
nameElement.addContent("dzy");
userElement.addContent(nameElement);
Element passwordElement = new Element("password");
passwordElement.addContent("6582254");
userElement.addContent(passwordElement);
Element true_nameElement = new Element("true_name");
true_nameElement.addContent("邓子云");
userElement.addContent(true_nameElement);
Element ageElement = new Element("age");
ageElement.addContent("26");
userElement.addContent(ageElement);
Element sexElement = new Element("sex");
sexElement.addContent("男");

```



```

userElement.addContent(sexElement);
Element addressElement = new Element("address");
addressElement.addContent("长沙市商业银行信息技术部");
userElement.addContent(addressElement);
Element telephoneElement = new Element("telephone");
telephoneElement.addContent("0731-4305524");
userElement.addContent(telephoneElement);
Element add_timeElement = new Element("add_time");
add_timeElement.addContent("2006-4-2 15:20:23");
userElement.addContent(add_timeElement);
//给 ageElement 元素创建名为 ageunit 的属性, 值为"岁"
ageElement.setAttribute(new Attribute("ageunit", "岁"));
//输出到控制台
Format format = Format.getPrettyFormat();
format.setEncoding("gb2312");//设置解码方式
XMLOutputter xmlOut = new XMLOutputter(format);
try {
    xmlOut.output(myDocument, System.out);
} catch (IOException e) {
    e.printStackTrace();
}
//输出到 XML 文件
FileWriter writer = new FileWriter("d:/user3.xml");
xmlOut.output(myDocument, writer);
writer.close();
%>
</body>
</html>

```

以上是本节创建 XML 文档的完整代码, 读者可自行调试看看运行的结果, 应当会在控制台输出 XML 文档, 并在 D 盘生成 user3.xml 文件。

#### 4. 读取与修改 XML 文件

JDOM 不光可以很方便地建立 XML 文档, 它的另一个用处是能够读取并操作现有的 XML 数据。

JDOM 的解析器在 org.jdom.input.\* 这个包里, 其中 DOMBuilder 的功能是将 DOM 模型的 Document 解析成 JDOM 模型的 Document; SAXBuilder 的功能是从文件或流中解析出符合 JDOM 模型的 XML 树。由于经常要从一个文件里读取数据, 因此应该采用后者作为解析工具。解析一个 xml 文档, 基本可以看成以下几个步骤。

##### ① 实例化一个合适的解析器对象。

```
SAXBuilder sb = new SAXBuilder();
```

##### ② 以包含 XML 数据的文件为参数, 构建一个文档对象 myDocument。

```
Document myDocument = sb.build("user.xml");
```

##### ③ 获取根元素。

```
Element rootElement = myDocument.getRootElement();
```

一旦获取了根元素, 就可以很方便地对它下面的子元素进行操作了, 下面对 Element 对象

的一些常用方法做一下简单说明：

(1) `getChild("childname")`: 返回指定名字的子节点，如果同一级有多个同名子节点，则只返回第一个；如果没有返回 `null` 值。

(2) `getChildren("childname")`: 返回指定名字的子节点 `List` 集合，这样就可以遍历所有的同一级同名子节点。

(3) `getAttributeValue("name")`: 返回指定属性名字的值，如果没有该属性则返回 `null`，有该属性但是值为空，则返回空字符串。

(4) `getChildText("childname")`: 返回指定子节点的内容文本值。

(5) `getText()`: 返回该元素的内容文本值。

以上 `getXxx()` 方法对应地就有 `setXxx()` 方法来修改元素中的文本值。

### 【实例 11-3】用 JDOM 方式读取 XML 文件

本实例将用 JDOM 读取 `user.xml` 文件并格式化显示。源代码如下：

JdomParseUserXML1.jsp

```
<%@ page contentType="text/html; charset=gb2312" language="java"%>
<%@ page import="java.io.*,org.jdom.*,org.jdom.input.*,
                org.jdom.output.*,java.util.List,
                java.util.Iterator" %>

<html>
<head><title>用 JDOM 解析并输出 user.xml</title></head>
<body>
<table border=1>
  <!--输出表头-->
  <tr>
    <td>用户 ID</td>
    <td>用户名</td>
    <td>密码</td>
    <td>真实姓名</td>
    <td>年龄</td>
    <td>性别</td>
    <td>联系地址</td>
    <td>联系电话</td>
    <td>加入系统时间</td>
  </tr>
<%//-----得到数据----->
  SAXBuilder builder = new SAXBuilder();//创建对象
  //建立 Document 对象
  Document readDocument = builder.build(
    pageContext.getServletContext().getResourceAsStream("/user.xml"));
  //得到根元素
  Element rootElement = readDocument.getRootElement();
  //得到根元素的子元素列表,实际上就是 user 元素列表
  List list = rootElement.getChildren();
  //-----输出数据-----
  for(Iterator i = list.iterator();i.hasNext();){
    Element current = (Element)i.next();
    out.println("<tr>");
```



```

//----输出用户 ID 号--
out.println("<td>"+current.getChildText("id")+"</td>");
//----输出用户名--
out.println("<td>"+current.getChildText("name")+"</td>");
//----输出用户密码--
out.println("<td>"+current.getChildText("password")+"</td>");
//----输出用户真实姓名--
out.println("<td>"+current.getChildText("true_name")+"</td>");
//----输出用户年龄--
out.println("<td>"+current.getChildText("age")+"</td>");
//----输出用户性别--
out.println("<td>"+current.getChildText("sex")+"</td>");
//----输出用户联系地址--
out.println("<td>"+current.getChildText("address")+"</td>");
//----输出用户联系电话--
out.println("<td>"+current.getChildText("telephone")+"</td>");
//----输出用户加入系统时间--
out.println("<td>"+current.getChildText("add_time")+"</td>");
out.println("</tr>");
}
%>
</table>
</body>
</html>

```

程序的运行结果如图 11-2 所示。

用户 ID	用户名	密码	真实姓名	年龄	性别	联系地址	联系电话	加入系统时间
1	dzy	6582254	邓子云	26	男	长沙市商业银行信息科技部	0731-4305524	2006-4-2 15:20:23

图 11-2 读取 XML 文件的内容

**提示** 使用 JDOM 读取 XML 文件时, 会自动根据<!DOCTYPE>语句验证 XML 文档是否符合 DTD 规范, 本例中的 user4.xml 将 user.xml 文件中的<!DOCTYPE>语句去掉了, 如果要打开 DTD 检验, 需要加入如下的语句:

```
<!DOCTYPE users SYSTEM "http://localhost:8080/ch11/user.dtd">
```

这样, XML 文件就能正确地找到 DTD 文件了。采用 XML Schema 校验方式亦如此。

## 11.2 上传和下载文件

HTML 表单中有 file 标签用于在客户端向服务器上传文件, 但接收客户端上传文件的服务器端程序代码需要 Java 程序员自行编写, 但编写这个程序是比较复杂的, 需要 Java 程序员掌

握更多的底层知识，为方便快捷应用，可以使用一些成熟的第三方组件，本节中将使用 `jspSmartUpload` 这一组件来轻松实现文件的上传处理。

使用 `jspSmartUpload` 组件相当简单，在 JSP 文件中仅仅书写三五行 Java 程序代码就可以实现文件的上传或下载，并能全程控制上传，利用 `jspSmartUpload` 组件提供的对象及其操作方法，可以获得全部上传文件的信息，如：文件名、大小、类型、扩展名及文件数据等，以方便文件的存取；能对上传的文件在大小、类型等方面做出限制，这样可以过滤掉不符合要求的文件。



**提示** 随书光盘中带有 `jspSmartUpload` 的组件包 `jspSmartUpload.jar`，位于本章的 Web 应用的“WEB-INF/lib”目录中。

### 11.2.1 jspSmartUpload 常用的 API

了解和掌握好 `jspSmartUpload` 组件相关的类是用好这个组件的前提，它主要有 4 个相关的类：`File` 类（并非 JDK 中的 `File` 类）、`Files` 类、`Request` 类和 `SmartUpload` 类。



**提示** 读者在阅读时也可以跳过本节 API 的说明而直接进入实例的学习，在程序中使用到 `jspSmartUpload` API 时再回到本节来查看相关属性、方法的说明。

#### 1. File 类

这个类包装了一个上传文件的所有信息。通过它可以得到上传文件的文件名、文件大小、扩展名及文件数据等信息。

##### (1) `saveAs()`

用来将文件另存为一个文件，如文件的换名保存。此方法的声明情况如下：

```
public void saveAs(java.lang.String destFilePathName, int optionSaveAs)
throws com.jspsmart.upload.SmartUploadException, java.io.IOException;
```

或：

```
public void saveAs(java.lang.String destFilePathName)
throws com.jspsmart.upload.SmartUploadException, java.io.IOException;
```

参数 `destFilePathName` 是另存为的文件名；参数 `optionSaveAs` 是另存为的选项，其值有 3 种，分别是 `SAVEAS_PHYSICAL`、`SAVEAS_VIRTUAL`、`SAVEAS_AUTO`。`SAVEAS_PHYSICAL` 表示以操作系统的根目录为文件根目录的另存文件，`SAVEAS_VIRTUAL` 表示以 Web 应用程序的根目录为文件根目录的另存文件，`SAVEAS_AUTO` 则表示让组件决定，当 Web 应用程序的根目录存在另存文件的目录时，它会选择 `SAVEAS_VIRTUAL`，否则会选 `SAVEAS_PHYSICAL`。如：

```
saveAs("/upload/test.doc", SAVEAS_PHYSICAL)
```

执行后若 Web 服务中间件软件安装在 C 盘，则另存的文件名实际是 `c:\upload\test.doc`。再如：

```
saveAs("/upload/test.doc", SAVEAS_VIRTUAL)
```

执行后若 Web 应用程序的根目录是 `webapps/ROOT`，则另存的文件名实际是 `webapps/`



ROOT/upload/test.doc。又如：

```
saveAs("/upload/test.doc", SAVEAS_AUTO)
```

执行时若 Web 应用程序根目录下存在 upload 目录，则其效果同 `saveAs("/upload/test.doc", SAVEAS_VIRTUAL)`，否则其效果与如下的语句相同：`saveAs("/upload/test.doc", SAVEAS_PHYSICAL)`。对于 Web 程序的开发来说，最好使用 `SAVEAS_VIRTUAL`，以方便程序的移植。

## (2) isMissing()

用于判断用户是否选择了文件，即提交的表单中对应的表单项是否有值，选择了文件时，返回 `false`，未选文件时，返回 `true`。

```
public boolean isMissing();
```

## (3) getFieldName()

得到 HTML 表单中对应于上传文件的表单项名字。

```
public java.lang.String getFieldName();
```

## (4) getFileName()

取文件名（不含目录信息）。

```
public java.lang.String getFileName();
```

## (5) getFilePathName()

取文件中带目录的全名。

```
public java.lang.String getFilePathName();
```

## (6) getFileExt()

取文件扩展名，即文件名的后缀。

```
public java.lang.String getFileExt();
```

## (7) getSize()

得到文件的长度，单位是字节。

```
public int getSize();
```

## (8) getBinaryData()

取文件数据中指定位移处的一个字节，用于检测文件处理。

```
public byte getBinaryData(int index);
```

参数 `index` 表示位移，其值在 0 到 `getSize()-1` 之间。

## 2. Files 类

`Files` 类表示所有上传文件的集合，通过它可以得到上传文件的数目、大小等信息。

### (1) getCount()

取得上传文件的数目。

```
public int getCount();
```

### (2) getFile()

取得指定位移处的文件对象 `File`，实际上是 `com.jspsmart.upload.File`，不是 `java.io.File`，请

读者注意区分两者之间的差别。调用方法：

```
public com.jspsmart.upload.File getFile(int index);
```

其中，参数 `index` 为指定的位移，其值在 0 到 `getCount()-1` 之间。

### (3) `getSize()`

取得上传文件的总长度，可用于限制一次性上传数据量的大小。

```
public int getSize()
```

### (4) `getCollection()`

将所有上传文件对象以 `Collection` 的形式返回，以便其他应用程序引用，浏览上传文件信息。

```
public java.util.Collection getCollection();
```

### (5) `getEnumeration()`

将所有上传文件对象以 `Enumeration`（枚举）的形式返回，以便其他应用程序浏览上传文件信息。

```
public java.util.Enumeration getEnumeration();
```

## 3. Request 类

`Request` 类的功能等同于 JSP 内置的对象 `request`。之所以提供这个类，是因为对于文件上传的表单，通过 JSP 内置的对象 `request` 对象无法获得表单中文件上传项的值，而必须通过 `jspSmartUpload` 组件提供的 `Request` 对象来获取。

`Request` 常用的方法有 `getParameter()`、`getarameterValues()`、`getParameterNames()`等，使用方法同 JSP 内置的对象 `request`。



**提示** `Request` 类的第 1 个字母为大写的“R”，不同于 JSP 内置的对象 `request`。

## 4. SmartUpload 类

这个类完成上传和下载的相关工作。

### (1) `initialize()`

这个方法是上传与下载共用的方法，用于进行上传下载的初始化工作，必须第一个执行。其使用形式有如下 3 种：

```
public final void initialize(javax.servlet.ServletConfig config,
    javax.servlet.http.HttpServletRequest request,
    javax.servlet.http.HttpServletResponse response)
    throws javax.servlet.ServletException;
```

或：

```
public final void initialize(javax.servlet.jsp.PageContext pageContext)
    throws javax.servlet.ServletException;
```

或：

```
public final void initialize(javax.servlet.ServletContext application,
    javax.servlet.http.HttpSession session,
    javax.servlet.http.HttpServletRequest request,
```



```
javax.servlet.http.HttpServletResponse response,
javax.servlet.jsp.JspWriter out)
throws javax.servlet.ServletException;
```

该方法无返回值，参数 `pageContext` 为 JSP 页面内置对象，即页面上下文；参数 `config`、`request`、`response`、`application`、`out` 分别可设为 JSP 相应的内置对象。以上的 3 种形式，第 2 种最为简单，推荐读者使用。

#### (2) upload()

此方法是上传文件时使用的方法，用于上传文件数据。对于上传操作，第 1 步执行 `initialize()` 方法，第 2 步就执行这个方法。

```
public void upload() throws com.jspsmart.upload.SmartUploadException,
java.io.IOException, javax.servlet.ServletException;
```

#### (3) save()

此方法是上传文件时使用的方法，方法将上传的全部文件保存到指定目录下，并返回保存的文件个数。

```
public int save(java.lang.String destPathName)
throws com.jspsmart.upload.SmartUploadException,
java.io.IOException, javax.servlet.ServletException;
```

或

```
public int save(java.lang.String destPathName, int option)
throws com.jspsmart.upload.SmartUploadException,
java.io.IOException, javax.servlet.ServletException;
```

参数 `destPathName` 为文件保存的目录，`option` 为保存选项，它有三个值，分别是 `SAVE_PHYSICAL`、`SAVE_VIRTUAL` 和 `SAVE_AUTO`。`SAVE_PHYSICAL` 表示组件将文件保存到以操作系统根目录为文件根目录的目录下；`SAVE_VIRTUAL` 表示组件将文件保存到以 Web 应用程序根目录为文件根目录的目录下；而 `SAVE_AUTO` 则表示由组件自动选择，这与 `File` 类的 `saveAs` 方法的选项值类似。

#### (4) getSize()

此方法是上传文件时使用的方法，用于得到上传文件数据的总长度。

```
public int getSize();
```

#### (5) getFiles()

此方法是上传文件时使用的方法，用于取得全部上传文件，以 `Files` 对象形式返回，从而可以利用 `Files` 类的操作方法来获得上传文件的数目等信息。

```
public com.jspsmart.upload.Files getFiles();
```

#### (6) getRequest()

此方法是上传文件时使用的方法，用于取得 `Request` 对象，以便由此对象获得上传表单参数的值。

```
public com.jspsmart.upload.Request getRequest();
```

#### (7) setAllowedFilesList()

此方法是上传文件使用的方法，用于设定允许上传带有指定扩展名的文件，当上传过程中

有文件名不允许时，组件将抛出异常。

```
public void setAllowedFilesList(java.lang.String allowedFilesList);
```

参数 `allowedFilesList` 为允许上传的文件扩展名列表，各个扩展名之间以逗号分隔。如果想允许上传那些没有扩展名的文件，可以用两个逗号表示。例如：`smartUploadObjectName.setAllowedFilesList("doc,txt,,")`将允许上传带.doc 和.txt 扩展名的文件以及没有扩展名的文件。

#### (8) setDeniedFilesList()

该方法是上传文件时使用的方法，用于限制上传带有指定扩展名的文件。若有文件扩展名被限制，则上传时组件将抛出异常。

```
public void setDeniedFilesList(java.lang.String deniedFilesList)
    throws java.sql.SQLException, java.io.IOException,
    javax.servlet.ServletException;
```

参数 `deniedFilesList` 为禁止上传的文件扩展名列表，各个扩展名之间以逗号分隔。如果想禁止上传没有扩展名的文件，可以用两个逗号来表示。例如：`smartUploadObjectName.setDeniedFilesList("exe,bat,,")`将禁止上传带.exe 和.bat 扩展名的文件以及没有扩展名的文件。

#### (9) setMaxFileSize()

该方法是上传文件时使用的方法，用于设定每个文件允许上传的最大长度。

```
public void setMaxFileSize(long maxFileSize);
```

参数 `maxFileSize` 为每个文件允许上传的最大长度，单位为字节，当文件超出此长度时，将不能被上传。

#### (10) setTotalMaxFileSize()

该方法是上传文件时使用的方法，用于设定允许上传文件的总长度，用于限制一次性上传的数据量大小。

```
public void setTotalMaxFileSize(long totalMaxFileSize);
```

参数 `totalMaxFileSize` 为允许上传文件的总长度，单位为字节。

#### (11) setContentDisposition()

该方法是下载文件时使用的方法，用于将数据追加到 MIME 文件头的 `CONTENT-DISPOSITION` 域中。`jspSmartUpload` 组件会在返回下载的信息时自动填写 MIME 文件头的 `CONTENT-DISPOSITION` 域，如果用户需要添加额外信息，请用此方法。

```
public void setContentDisposition(java.lang.String contentDisposition);
```

其中，参数 `contentDisposition` 为要添加的数据。如果 `contentDisposition` 为 `null`，则组件将自动添加“`attachment;`”，以表明将下载的文件作为附件，结果是 IE 浏览器将会提示“另存为”文件，而不是自动打开这个文件。IE 浏览器一般根据下载的文件扩展名决定执行的操作，如扩展名为.doc 将用 Word 程序打开，扩展名为.pdf 将用 acrobat 程序打开，当然前提是客户端要安装有相应的应用软件。

#### (12) downloadFile()

该方法是下载文件时使用的方法，用于下载文件。

```
public void downloadFile(java.lang.String sourceFilePathName)
    throws com.jspsmart.upload.SmartUploadException,
```



```
java.io.IOException, javax.servlet.ServletException;
```

或

```
public void downloadFile(java.lang.String sourceFilePathName,
    java.lang.String contentType) throws com.jspsmart.upload.SmartUploadException,
    java.io.IOException, javax.servlet.ServletException;
```

或

```
public void downloadFile(java.lang.String sourceFilePathName,
    java.lang.String contentType, java.lang.String destFileName)
    throws com.jspsmart.upload.SmartUploadException, java.io.IOException,
    javax.servlet.ServletException;
```

其中第 1 种方式最常用，后两种用于特殊情况下的文件下载，如：更改内容类型、更改另存为的文件名。

参数 `sourceFilePathName` 为要下载的文件名，是带目录的文件全名；参数 `contentType` 为内容类型（MIME 格式的文件类型信息，可被浏览器识别）；参数 `destFileName` 为下载后默认的另一存为的文件名。

### 11.2.2 上传文件

本节中将用一个上传文件的实例来讲解具体是如何利用 `jspSmartUpload` 组件实现文件上传功能的。

#### 【实例 11-4】上传文件

假设现在要做一个网上书店后台管理中的录入书籍信息功能，其中就要录入书籍的封面。这就需要有一个页面来建立一个录入书籍相关信息的表单，这个表单中就有录入封面图片的文件控件，录入完后，提交要接收页面，在接收页面中利用 `jspSmartUpload` 将文件上传到服务器。

uploadFileForm.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head><title>上传文件</title></head>
<body>
<form METHOD="POST" ACTION="uploadFileAccept.jsp" NAME="book"
    ENCTYPE="multipart/form-data">
    <table CELLSPACING="0" CELLPADDING="3" BORDER="1" WIDTH="474">
        <tr><td align="center" colspan="2">录入书籍信息</td></tr>
        <tr>
            <td width="150">书名:</td>
            <td width="324"><input TYPE="TEXT" name="bookname" ></td>
        </tr>
        <tr>
            <td>封面:</td>
            <td><input TYPE="file" name="photofile"></td>
        </tr>
        <tr>
            <td>作者:</td>
```

```

        <td><input TYPE="TEXT" name="author"></td>
    </tr>
    <tr>
        <td>出版社:</td>
        <td><input TYPE="TEXT" name="publisher"></td>
    </tr>
    <tr><td colspan="2" width="474">
        <input TYPE="Submit" value="提交"></td>
    </tr>
</table>
</form>
</body>
</html>

```

程序运行结果如图 11-3 所示。

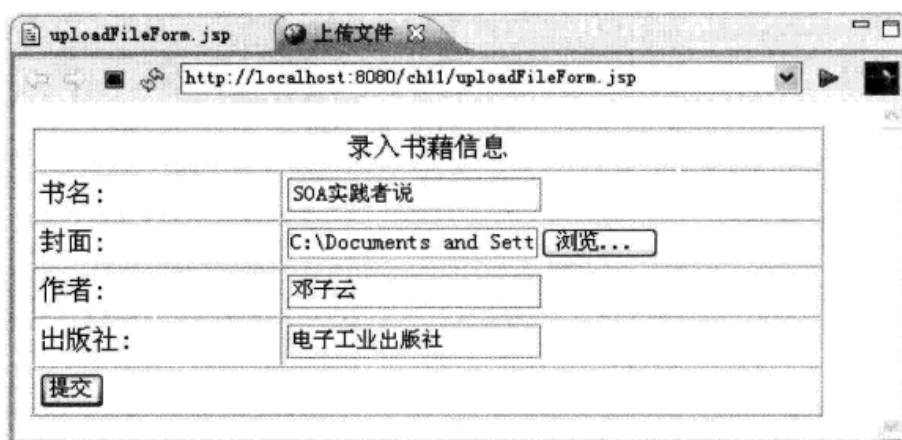


图 11-3 录入书籍信息页面

在编写代码时，需要注意如果表单要用来上传文件，则须将表单<form>标签的 ENCTYPE 属性值设为“multipart/form-data”。在这个页面中输入信息后，数据被提交到 uploadFileAccept.jsp 页面，代码如下：

#### uploadFileAccept.jsp

```

<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ page import="com.jspsmart.upload.*"%>
<html>
<head><title>上传文件</title></head>
<body>
<%
//新建一个 SmartUpload 对象
SmartUpload su = new SmartUpload();
//上传初始化,pageContext 为 JSP 的内置对象
su.initialize(pageContext);
//设定允许上传的文件（通过扩展名限制），仅允许 jpg,bmp,gif 文件。
su.setAllowedFilesList("jpg,bmp,gif");
//上传文件
su.upload();
//将上传文件全部保存到指定目录
su.save("upload", SmartUpload.SAVE_VIRTUAL);

```



```

%>
<table CELSPACING="0" CELLPADDING="3" BORDER="1" WIDTH="474">
  <tr>
    <td width="150">书名:</td>
    <td width="324">
      <%=su.getRequest().getParameter("bookname")%>
    </td>
  </tr>
  <tr>
    <td>封面:</td>
    <td>"></td>
  </tr>
  <tr>
    <td>作者:</td>
    <td><%=su.getRequest().getParameter("author")%></td>
  </tr>
  <tr>
    <td>出版社:</td>
    <td><%=su.getRequest().getParameter("publisher")%></td>
  </tr>
</table>
</body>
</html>

```

程序运行的结果如图 11-4 所示。



图 11-4 上传图片

从以上程序代码可以看出, 仅用了 5 行代码就实现了文件上传的功能。如果要得到表单中的数据项, 需要使用 `jspSmartUpload` 组件的 `Request` 对象, 而不能使用 JSP 的内容对象 `request`。

为存放上传的文件, 需要事先在服务器中创建文件夹来存放, 本例就事先在 Web 应用的目录中创建了 `upload` 文件夹。请读者注意这一点, 否则会上传失败。但本例中的方法并不安全,

因为访问者可以通过猜测上传的文件所在的路径及文件的名称, 直接通过 URL 地址访问上传的文件。为安全起见, 有两种解决办法: 一是将存放上传文件的文件夹放置于 Web 应用的“WEB-INF”文件夹中, 因为这个文件夹是在客户端不能访问的; 二是在服务器中另建一个文件夹, 专门用于存放上传的文件, 但是这样损失了 Web 应用的可移植性。

有时, 也可能需要在一个表单中上传多个文件, 比如本例中, 一本图书的封面有首封和底封, 则就需要一张首封的图片, 一张底封的图片, 这就需要一次上传 2 个文件, 这又怎么办呢? 通过 SmartUpload 类的 `getFiles().getFile(0).getFileName()` 方法即可得到上传的文件名, 其中 `getFile()` 方法的参数为索引号, 从 0 开始, 即表单中的第 1 个文件框对应的索引号为 0, 第 2 个为 1, 以此类推。上传文件的动作则通过 SmartUpload 类 `upload()` 方法就全部上传完毕了。

### 【实例 11-5】上传文件到数据库

使用 `jspSmartUpload` 组件可以很轻松地将文件上传到服务器中, 那么能否将图片上传到数据库中呢? 这样客户端就不能通过 URL 直接访问上传的文件, 而需要从数据库中读取, 增强了安全性, 但同时也降低了效率, 因为从数据库中输出数据相对较慢。

文件的内容存放到数据库中, 实际上是作为二进制方式存储的, 在 SQL Server 中 `image` 型的字段可以用来存放文件的内容。本例将参照实例 11-4, 继续延用这个上传书籍图片的实例。首先需要在数据库中建表, 建表的 SQL 语句如下:

```
create table book(bookId int identity(1,1),
    bookName varchar(80),bookImage image,
    author varchar(80),publisher varchar(40))
```

上传文件的 JSP 页面代码如下。

#### uploadFile1.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<html>
<head><title>上传文件</title></head>
<body>
<form METHOD="POST" ACTION="uploadFile2.jsp" NAME="book"
    ENCTYPE="multipart/form-data">
    <table CELLSPACING="0" CELLPADDING="3" BORDER="1" WIDTH="474">
        <tr><td align="center" colspan="2">录入书籍信息</td></tr>
        <tr>
            <td width="150">书名:</td>
            <td width="324"><input TYPE="TEXT" name="bookname" ></td>
        </tr>
        <tr>
            <td>封面:</td>
            <td><input TYPE="file" name="photofile"></td>
        </tr>
        <tr>
            <td>作者:</td>
            <td><input TYPE="TEXT" name="author"></td>
        </tr>
        <tr>
            <td>出版社:</td>
```



```

        <td><input TYPE ="TEXT" name="publisher"></td>
    </tr>
    <tr><td colspan="2" width="474">
        <input TYPE="Submit" value="提交"></td>
    </tr>
</table>
</form>
</body>
</html>

```

运行结果同图 11-3。其实代码也大致和实例 11-4 中的 uploadFileForm.jsp 代码相似，仅修改了<form>标签中的 action 属性的值，将数据提交到 uploadFile2.jsp 页面。

#### uploadFile2.jsp

```

<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ page import="com.jspsmart.upload.*,java.sql.*,java.io.*"%>
<html>
<head><title>上传文件</title></head>
<body>
<%//-----将上传的图片保存为一个文件----->
//新建一个 SmartUpload 对象
SmartUpload su = new SmartUpload();
//上传初始化,pageContext 为 JSP 的内置对象
su.initialize(pageContext);
//设定允许上传的文件（通过扩展名限制），仅允许 jpg,bmp,gif 文件。
su.setAllowedFilesList("jpg,bmp,gif");
//上传文件
su.upload();
//将上传文件全部保存到指定目录
su.save("upload",SmartUpload.SAVE_VIRTUAL);
%>
<%//-----将上传的图片放入到数据库表的对应字段中----->
String filename=request.getSession().getServletContext().getRealPath("upload")+
"/"+su.getFiles().getFile(0).getFileName();
Connection con=null;
con=DriverManager.getConnection("jdbc:sqlserver://localhost:1433;
DatabaseName=testDatabase","sa","123");
FileInputStream bookImageFile=new FileInputStream(filename);
String sql="insert into book(bookName,bookImage,author,publisher)
values(?,?,?,?)";
PreparedStatement pstmt=con.prepareStatement(sql);
pstmt.setString(1,su.getRequest().getParameter("bookname"));
pstmt.setBinaryStream(2,bookImageFile,bookImageFile.available());
pstmt.setString(3,su.getRequest().getParameter("author"));
pstmt.setString(4,su.getRequest().getParameter("publisher"));
pstmt.execute();
%>
<table CELLSPACING="0" CELLPADDING="3" BORDER="1" WIDTH="474">
    <tr>
        <td width="150">书名:</td>
        <td width="324">
            <%=su.getRequest().getParameter("bookname")%>
        </td>
    </tr>

```

```

</tr>
<tr>
  <td>封面:</td>
  <td>&author=<%=su.getRequest().getParameter("author")%>
    &publisher=<%=su.getRequest().getParameter("publisher")%>"></td>
</tr>
<tr>
  <td>作者:</td>
  <td><%=su.getRequest().getParameter("author")%></td>
</tr>
<tr>
  <td>出版社:</td>
  <td><%=su.getRequest().getParameter("publisher")%></td>
</tr>
</table>
</body>
</html>

```

这个页面的运行结果如图 11-5 所示。



图 11-5 上传文件到数据库

从程序代码来看，先将文件上传到服务器的一个目录中，再利用 SQL 操作将文件数据插入到数据库中。显示图片则使用了一个 Servlet，这个 Servlet 的代码如下。

#### ShowImage.java

```

package file;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.sql.Connection;
import java.sql.DriverManager;

```



```

import java.sql.ResultSet;
import java.sql.Statement;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;
public class ShowImage extends HttpServlet {
    public void service(ServletRequest request, ServletResponse response)
        throws ServletException, IOException {
        try{
            Connection conn=
            DriverManager.getConnection("jdbc:sqlserver://localhost:1433;
                DatabaseName=testDatabase","sa","123");
            Statement stmt = conn.createStatement();
            String sql="select bookImage from book where
                bookName='"+codeToString(request.getParameter("bookname"))+
                "' and author='"+codeToString(request.getParameter("author"))+
                "' and publisher='"+
                codeToString(request.getParameter("publisher"))+"'";
            ResultSet rs = stmt.executeQuery(sql);
            if(rs.next()) {
                int len = 10 * 1024 * 1024;
                InputStream in = rs.getBinaryStream("bookImage");
                OutputStream toClient = response.getOutputStream();
                response.reset(); //返回在流中被标记过的位置
                response.setContentType("image/jpg"); //或 gif 等
                byte[] P_Buf = new byte[len];
                int i;
                while ((i = in.read(P_Buf)) != -1) {
                    toClient.write(P_Buf, 0, i);
                }
                in.close();
                toClient.flush(); //强制清出缓冲区
                toClient.close();
            }
            conn.close();
        }catch(Exception e){
            System.out.println("出现异常: " + e.getMessage());
        }
    }
    //处理中文字符串的函数
    public String codeToString(String str){
        return (new String(s.getBytes("ISO-8859-1")))
    }
}

```

Servlet 还需要部署, 本例中在 WEB-INF 文件夹下的 web.xml 中加入了如下的配置:

```

<servlet>
    <servlet-name>ShowImage</servlet-name>
    <servlet-class>file.ShowImage</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ShowImage</servlet-name>
    <url-pattern>/ShowImage</url-pattern>
</servlet-mapping>

```

### 11.2.3 下载文件

下载文件功能的实现其实是比较简便的，最为简单的方式就是使用超链接，链接的目标地址 URL 就是要下载的文件，如果是使用这种方法的话，浏览器会自动根据文件的扩展名在客户机上查找对应的应用程序来打开文件，如扩展名为.doc 则用 word 打开；如果找不到对应的应用程序则弹出一个“文件下载”对话框，即可将文件保存到客户机中。也可以运用 jspSmartUpload 组件来实现文件下载，则可以根据需要来控制下载。

#### 【实例 11-6】下载文件

downloadFile.jsp

```
<%@ page contentType="text/html; charset=gb2312" import="com.jspsmart.upload.*" %>
<%
    //新建一个 SmartUpload 对象
    SmartUpload su = new SmartUpload();
    //初始化
    su.initialize(pageContext);
    //设定 contentDisposition 为 null 以禁止浏览器自动打开文件，保证单击链接后是下载文件
    //若不设定，则下载的文件扩展名为 doc 时，浏览器将自动用 word 打开它。扩展名为 pdf 时
    //浏览器将用 acrobat 打开
    su.setContentDisposition(null);
    //下载文件
    su.downloadFile("upload/11.rar");
    //处理输出流问题
    out.clear();
    out = pageContext.pushBody();
%>
```

程序运行后会弹出一个“文件下载”对话框，如图 11-6 所示。

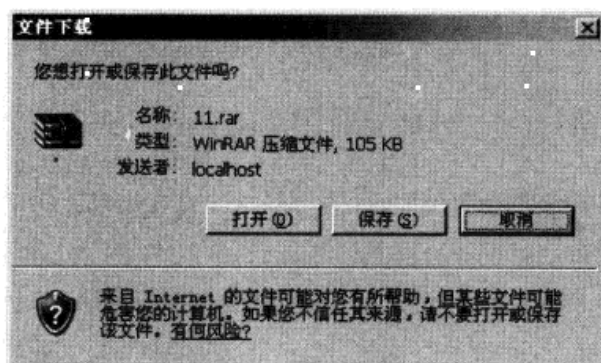


图 11-6 下载文件

单击“保存”按钮，再在接下来的“另存为”对话框中设置下载后的文件名，即可下载文件。

JSP 文件的最后用了 2 条语句来“处理输出流问题”，这是怎么回事呢？如果没有这 2 条语句，浏览器会报出以下错误：

```
getOutputStream() has already been called for this response
```



原因是 `su.initialize(pageContext)` 语句使用了输出流，而在 Tomcat 中 JSP 编译成 Servlet 之后，在函数 `_jspService(HttpServletRequest request, HttpServletResponse response)` 的最后有一段这样的代码：

```
finally {  
    if (_jspxFactory != null) _jspxFactory.releasePageContext  
        (_jspx_page_context);  
}
```

这里是释放在 JSP 中使用的对象，会调用 `response.getWriter()`，因为这个方法和 `response.getOutputStream()` 相冲突，所以会出现以上那个异常。解决问题的办法是，在使用 `OutputStream` 输出流完成后，调用下面的 2 个方法即可解决该问题。

```
out.clear();  
out = pageContext.pushBody();
```

## 11.3 制作 Web 报表与图形

制作 Web 报表与图形的方法有很多，简单的情况下可以在 JSP 页面中直接生成报表，需要打印时就用 JavaScript 的 `window.print()` 方法打印当前网页中的内容，然而开发人员又总觉得不美观；图形也可以使用 Java 编程来实现，但需要开发人员掌握较多的底层 API，且代码可能比较冗长。

为简便起见，可以直接使用一些第三方组件来快速开发，如 JFreeChart、jasperreport 等，本节中介绍一款国产的第三方组件——JavaReport，它既可以用来开发 Web 报表，也能用来制作 Web 图形。



**提示** 随书光盘中带有 JavaReport 的组件包 `JavaReport-V3-Enterprise-Released.jar`，位于本章的 Web 应用的“WEB-INF/lib”目录中。

### 11.3.1 JavaReport 简介

JavaReport 主要有如下优点。

#### 1. 支持实时的、动态的 Web 统计报表

JavaReport 所展示的报表是动态的报表。所有的数据都是实时的数据；所有的统计图也是动态生成的。报表里面的数据源可以从数据库中即时查询产生，适合于对实时统计要求高的应用系统。

#### 2. 接口丰富，对图表提供良好的支持

JavaReport 提供了丰富的应用程序接口、减少代码复用、灵活的图表形式、即时图表生成、更短的生成时间等。Java 报表对图表有良好的支持，可以生成柱图、饼图和折线图等十几种图表。

### 3. 报表可导出成各种格式的文档

JavaReport 在报表展现方面表现优异，能实现 Word，Excel，PDF，CSV 等格式精确导出等功能，并提供了全部的页面与打印控制。

### 4. 开发简便，程序员只须编写少量程序代码

有许多问题在 JavaReport 组件内已经解决，如报表的导出功能、打印功能和数据的分页处理等。在很多 B/S 结构体系的系统开发时，分页是开发过程中要重点考虑的问题。但在 JavaReport 中，就不需要考虑上下翻页，跨页分页，因为这些问题报表引擎已经实现了。在设计报表的过程中，把 Report 类当作容器类、统计图类、表格类、标签类、图片类等，把它们当作组件类，至于跨页分页、上下翻页由报表引擎自动完成。JavaReport 系统中有自动跨页分页的功能，当表格超过当前页的大小时，系统会自动把接着的部分放到下一页。报表设计在开发过程中感觉不到跨页的存在，只有一个全部数据完整的表格对象。

如果把引入图片文件加入到报表中，可使用报表系统的 Report 类，通过这个类的 addImage() 方法把 Image 对象加进来，如果需要自定义图像，可通过第三方画图程序（例如是 JFreeChart 程序）生产需要的统计图/图片，然后再把图像加到报表中去。

本节中使用的 JavaReport 版本是 V3.0，JavaReport 根据客户的应用需要，按照功能和性能的等级，分成三个版本：专业版，服务器版，企业版。本书使用企业版。

JavaReport 三个版本产品都是免费使用的。也就是说这三个版本都不需要购买 License 许可，可以自由使用 JavaReport 的所有功能。免费提供报表在 Web 展示功能，表和图混合功能，跨页分页功能，导出电子文档 Word、PDF、Excel、CSV、HTML 等诸多功能。

专业版客户（Client）的 IP 最大连接数限制为 10 个，也就是 JavaReport 同时并行处理线程的最大数受限制。限制是为了避免系统资源占用过大，使系统反应迟钝。该版本适合中小型的应用系统，保障应用系统正常运行。

服务器版的最大连接数没有受限，它能充分发挥服务器的各部分硬件设备的作用；相对要求服务器的设备配置高，保障最优性能效率。该版本适合大中型的应用系统使用。

企业版提供垃圾回收机制管理，自动处理在内存中无用对象的引用；支持多服务器处理模式，提供在多个服务器的集群功能和服务器之间负载均衡功能。提高服务器群的负载能力和快速响应能力。该版本适合负载繁重的应用系统使用。

## 11.3.2 JavaReport 常用的 API

要了解 JavaReport 常用的 API，就需要先了解用 JavaReport 制作了的 Web 报表显示时的情况，这样才会更加形象地理解这些 API。Web 报表显示时的情况如图 11-7 所示，报表的组成情况一目了然。



**提示** 读者在阅读时也可以跳过本节 API 的说明而直接进入实例的学习，在程序中使用到 JavaReport API 时再回到本节来查看相关属性、方法的说明。



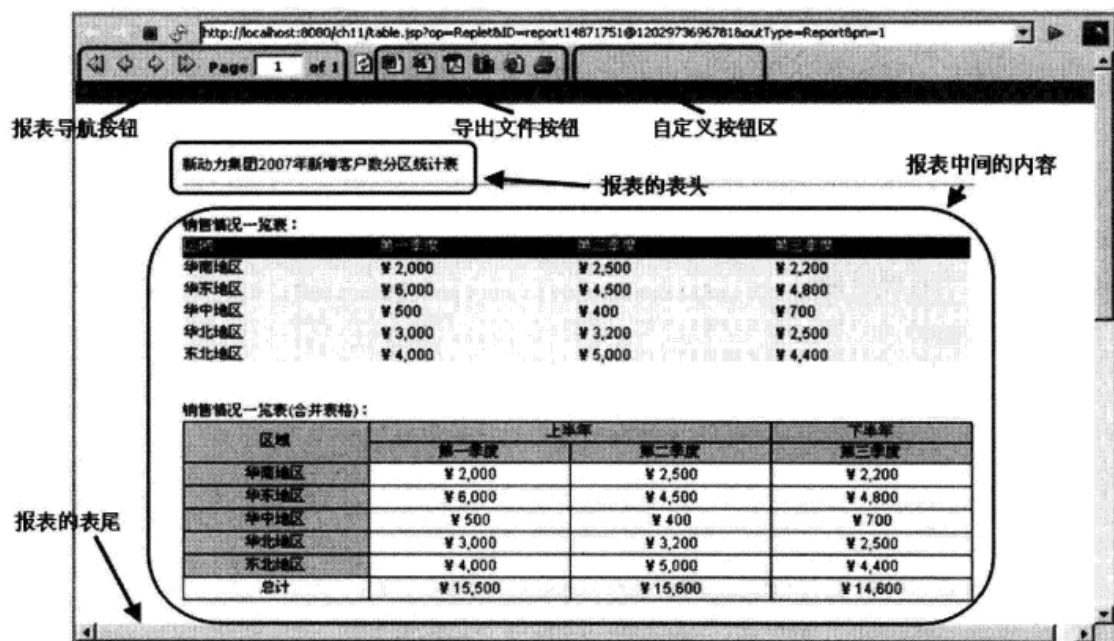


图 11-7 Web 报表显示时的情况

### 1. WebReportEngine 类

WebReportEngine 类即 `com.javareport.http.WebReportEngine`, 这个类是整个 JavaReport 中的 Web 引擎的开发接口。所有的 JSP 或 Servlet 从这个类继承下来, 覆盖 `createReport()` 函数就实现基本的报表开发工作。

WebReportEngine 是 JavaReport 的 Servlet 或 JSP 的开发接口, 是 JSP 或 Servlet 的父类, 报表系统在应用中的程序从此处继承下来, 可以根据需要调整接口内容, 部分函数可以适当删除。在开发中, 一般情况下实现 `createReport()` 函数就可以了, 形成实时动态报表就是在这个函数里实现的。剩下的工作 (怎样在 Web 上显示, 怎样形成 Word、PDF 文件等) 交给报表引擎自动实现。

如果是在 JSP 中, 在 JSP 页面的首部分应加入如下语句:

```
<%@ page extends="com.javareport.http.WebReportEngine"%>
```

如果是 Servlet, 相应的 Servlet 类声明语句如下:

```
public class Servlet 类名称 extends WebReportEngine{
    .....
}
```

#### (1) createReport()

方法原型如下:

```
public Report createReport(javax.servlet.http.HttpServletRequest request)
    throws java.lang.Exception
```

此方法用于建立报表, 并返回报表的实例。这个报表实例可以在 Web 上显示, 同时也可以导出 Word、Excel、PDF、CSV、HTML 等格式的文档以供使用。参数 `request` 可用于获取所有的动态请求的数据。

## (2) validate()

此方法用于对上一个页面中的表单提交的参数进行检查,实时报表需要动态的参数,在这个方法中可以进行数据校验。方法原型如下:

```
public java.lang.String validate(javax.servlet.http.HttpServletRequest request)
```

参数 `request` 可用于获取所有的动态请求的数据;方法的返回值为 `null` 时代表通过,其他内容则为参数错误的提示信息。

## (3) getStartScript()

此方法用于构造报表内容在 Web 页面上显示之前执行的 JavaScript 或 VBScript 脚本,如果要定制则要重载这个方法。方法原型如下:

```
public java.lang.String getStartScript(  
    javax.servlet.http.HttpServletRequest request)
```

参数 `request` 可用于获取所有动态请求的数据;方法返回值为 `null` 时代表没有脚本内容。

## (4) getEndScript()

此方法用于构造报表内容在 Web 页面上显示之后执行的 JavaScript 或 VBScript 脚本,如果要定制则要重载这个方法。方法原型如下:

```
public java.lang.String getEndScript(  
    javax.servlet.http.HttpServletRequest request)
```

参数 `request` 可用于获取所有动态请求的数据;方法返回值为 `null` 时代表没有脚本内容。

## (5) getToolBarScript()

此方法用于定制 Web 报表在页面首部显示的工具栏为标准的样式(上下翻页,导出文件),可以在此扩展工具栏的内容,如:添加公司主页的链接,返回上一层链接的“返回”按钮,如果要定制则要重载这个方法。方法原型如下:

```
public java.lang.String getToolBarScript(  
    javax.servlet.http.HttpServletRequest request)
```

参数 `request` 可用于获取所有动态请求的数据;方法返回值为 `null` 时代表不添加内容。

## (6) isShowToolBar()

如果不想在页面上显示报表工具栏,可重载这个方法,并设定返回值为 `false`。需要注意的是,如果是多页报表,上下翻页按钮就无法使用。方法原型如下:

```
public boolean isShowToolBar()
```

此方法的返回值是是否显示工具栏的布尔值标志。

## (7) getAllEchoButton()

此方法用于自定义显示在 Web 页面中的报表导出文件的按钮,比如应用中只导出 PDF 文件,其他的不需要,就可以这里设定。按钮值从 Word 按钮开始是 (1, 2, 4, 8, ...), 需要显示的按钮则将它们值相加就可以了。默认情况下工具栏上的按钮如图 11-8 所示:

```
public int getAllEchoButton()
```

此方法的返回值是显示按钮对应的和值。要作自定义,需要重载这个方法,并将返回值设为要显示的按钮对应的和值。



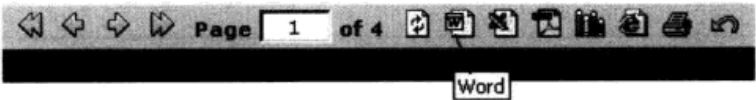


图 11-8 默认情况下工具栏上的按钮

2. Report 类

Report 类即 `com.javareport.beans.Report`，报表类。这个类的对象代表一张报表，是所有报表元素的容器。

报表的表头操作常用方法如表 11-2 所示。

表 11-2 报表的表头操作常用方法

方法名称	方法说明
<code>addHeaderSeparator(int num)</code>	报表的页眉添加一条横直线，参数 <code>num</code> 表示横直线的粗细程度，以自然数表示，数字越大表示线越粗
<code>addHeaderSpace(int num)</code>	在页眉中添加若干个空格，参数 <code>num</code> 是要添加的空格的个数
<code>addHeaderTab(int num)</code>	在页眉中添加若干个 Tab 键，Tab 键跟空格键一样是不可见的，默认每个 Tab 为 8 个空格。参数 <code>num</code> 是要添加的 Tab 键的个数
<code>addHeaderBreak()</code>	在页眉中添加一个换行符号，紧跟后面的内容则从下行第一个字符的位置开始
<code>addHeaderText(String text)</code>	在页眉中添加文本信息内容，紧跟后面的内容则从下行第一个字符的位置开始。参数 <code>text</code> 是要添加的文本信息内容。其中， <code>{P}</code> 代表当前页， <code>{N}</code> 代表总页数，如：“第{P}页，共{N}页”

报表中间的内容操作常用方法如表 11-3 所示。

表 11-3 报表中间的内容操作常用方法

方法名称	方法说明
<code>addChart(ChartImpl chart)</code>	在报表中添加图表信息内容。参数 <code>chart</code> 是要添加的图表
<code>setCurrentFont(java.awt.Font font)</code>	设置报表当前的字体。参数是要设置的字体对象，为 Java 中 <code>java.awt.*</code> 包中的 <code>Font</code> 对象
<code>setCurrentBackground(java.awt.Color color)</code>	设置报表当前的背景颜色。参数 <code>color</code> 是要添加的图形对象，为 Java 中 <code>java.awt.*</code> 包中的 <code>color</code> 对象
<code>setCurrentForeground(java.awt.Color color)</code>	设置报表当前的前景颜色。参数 <code>color</code> 是要添加的图形对象，为 Java 中 <code>java.awt.*</code> 包中的 <code>color</code> 对象
<code>addImage(java.awt.Image image)</code>	在报表中添加图片。参数 <code>image</code> 是要添加的图形对象，为 Java 中 <code>java.awt.*</code> 包中的 <code>Image</code> 对象
<code>addBullet()</code> 或 <code>addBullet(java.awt.Image image)</code>	添加项目符号的表示符号。第 1 种形式添加默认的项目符号（圆点）的表示符号，第 2 种形式用自定义的图片内容代替默认的圆点内容。参数 <code>image</code> 是要添加的图形对象，为 Java 中 <code>java.awt.*</code> 包中的 <code>Image</code> 对象
<code>addNewline(int num)</code>	在报表中添加换行符号，紧跟后面的内容则从下一行第一个字符的位置开始。参数 <code>num</code> 代表换行的次数
<code>addPageBreak()</code>	用于在报表中添加换页的标识符号，当报表系统做分页显示时，遇到这个符号时，则紧跟后面的内容在新的一页开始显示

续表

方法名称	方法说明
addSeparator(int num)	在报表中添加横直线。参数 num 表示横直线的粗细程度，以自然数表示，数字越大表示线越粗
addSpace(int num)	添加若干个空格。参数 num 代表空格的个数
addTab(int num)	在报表中添加若干个 Tab 键。参数 num 是要添加的 Tab 键的个数
addText(String text)	在报表中添加文本信息内容，紧跟后面的内容则从下一行第一个字符的位置开始。参数 text 为要添加的文本信息内容
addTable(Table table)	添加报表中的表格信息内容。参数 table 是要添加的表格

报表表尾的内容操作常用方法如表 11-4 所示。

表 11-4 报表表尾的内容操作常用方法

方法名称	方法说明
addFooterSeparator(int num)	在页尾添加一条横直线。参数 num 为横直线的粗细程度，以自然数表示，数字越大表示线越粗
addFooterSpace(int num)	在页尾添加若干个空格。参数 num 代表要添加的空格的个数
addFooterTab(int num)	在页尾添加若干个 Tab 键。参数 num 为要添加的 Tab 键的个数
addFooterBreak()	在页尾添加一个换行符号，紧跟后面的内容则从下一行第一个字符的位置开始
addFooterText(String text)	在页尾添加文本信息内容，紧跟后面的内容则从下一行第一个字符的位置开始。参数 text 是要添加的文本信息内容。其中，{P}代表当前页，{N}代表总页数，如：“第{P}页，共{N}页”

3. Table 类

Table 类即 com.javareport.beans.Table，表格类。这个类的对象属于报表对象 Report 中的元素。表格在报表中是不可缺少的，整齐排列着数据内容。表格单元里面的内容可以是文本内容，也可以是图形和其他元素，同时这个对象也是表格套表格的基础。JavaReport 会自动处理表格的跨页、分页问题和新页中的表头显示问题，开发过程中把它想象成连续的就可以了。

Table 类的常用方法如表 11-5 所示。

表 11-5 Table 类的常用方法

方法名称	方法说明
Table(java.lang.Object[][] data)	构造函数，用于实例化一个表格对象。参数 data 为填充表格内容的二维数组
setRowAlignment(int row,int align)	设置指定行的对齐方式。参数 row 为要设置对齐方式的行编号；参数 align 为要对齐的方式（左，中，右；上，中，下；同时设置左右上下，将两个值相加就可以了）
setRowBackground(int row,java.awt.Color color)	设置指定行的背景颜色。参数 row 为要设置背景颜色的行编号；参数 color 为要设置的颜色对象，为 Java 中 java.awt.*包中的 Color 对象
setRowBorder(int style)	设置表格中所有行的边界外观样式。参数 style 为边界外观样式，即表格单元之间的边界线的粗细情况，该值为整数，值越大代表线越粗，0 代表不显示



续表

方法名称	方法说明
setRowBorderColor(int row,java.awt.Color color)	设置表格中指定行的边界的颜色,参数 row 为要设置边界颜色的行编号;参数 color 是要设置的颜色对象,为 Java 中 java.awt.*包中的 Color 对象
setRowFont(int row,java.awt.Font font)	设置表格中指定行内容的字体。参数 row 为要设置字体的指定行;参数 font 是要设置的字体对象,为 Java 中 java.awt.*包中的 Font 对象
setRowForeground(int row,java.awt.Color color)	设置表格中指定行的前景颜色。参数 row 是要设置前景颜色的行编号;参数 color 是要设置的颜色对象。
setRowHeight(int row,int height)	设置表格中指定行的高度。参数 row 是要设置高度的行编号;参数 height 是要设置的行的高度
setColAlignment(int col,int align)	设置表格中指定列的对齐方式。参数 col 是指定对齐方式的列;参数 align 指出对齐的方式,依次为横向左,中,右;竖向上,中,下(如果要同时设置左右上下,可将两个值相加),值可以用数字表示,也可以用 Table 类的常量来表示
setColBackground(int index,java.awt.Color color)	设置表格中指定列的背景颜色。参数 index 是指定的列;参数 color 是要设置的颜色对象
setColBorder(int style)	设置表格的所有列的边界外观样式。参数 style 指定边界外观样式,即表格单元之间边界线的粗细情况,该值为整数,值越大代表线越粗,0 代表不显示
setColBorder(int index,int style)	设置表格中指定列的边界外观样式。参数 int 为指定的列
setColBorderColor(int index,java.awt.Color color)	设置表格中指定列的边界的颜色。参数 index 为指定的列;参数 color 是要设置的颜色对象
setColFont(int index,java.awt.Font font)	设置表格中指定列内容的字体。参数 index 为指定的列;参数 font 是要设置的字体对象
setColForeground(int index,java.awt.Color color)	设置表格中指定列的前景颜色。参数 index 为指定的列;参数 color 是要设置的颜色对象
setColWidth(int index,int width)	设置表格中指定列的宽度。参数 index 是要设置高度的列编号;参数 width 为要设置的列的宽度
setData(java.lang.Object[][] data)	给表格填充数据。表格单元的数据可以是数据,也可以是其他元素。参数 data 是填充表格内容的二维数据
setAlignment(int align)	设置表格中所有单元的对齐方式。参数 align 是要对齐的方式
setColAutoSize(boolean flag)	设置表格中所有列是否自动调整宽度。参数 flag 是布尔型的标志值
setRowAutoSize(boolean flag)	设置表格中所有行是否自动调整宽度。参数 flag 是布尔型的标志值
setFont(java.awt.Font font)	设置所有表格单元的字体。参数 font 是要设置的字体对象
setHeaderColCount(int count)	设置表格列表头的数目。参数 count 为列表头的数目
setHeaderRowCount(int count)	设置表格行表头的数目。参数 count 为行表头的数目
setRowHeight(int height)	设置行的默认高度。参数 height 为默认的高度值
setCellSpan(intx,int y,java.awt.Dimension dimension)	合并表格单元,即把连续的多个表格单元和并成为一个表格。参数 x 为合并表格左上角表格的坐标 x 值;参数 y 为合并表格左上角表格的坐标 y 值;参数 dimension 为合并的矩形框的大小,即跨越表格的面积
setLineWrap(boolean flag)	设置表格单元的内容超宽时是否换行显示。参数 flag 是布尔型的标志值
setFont(int x,int y,java.awt.Font font)	设置指定表格单元内容的字体。参数 x 是要设置字体表格单元的坐标 x 值;参数 y 是要设置字体表格单元的坐标 y 值;参数 font 是要设置的字体对象

续表

方法名称	方法说明
<code>setForeground(int x,int y,java.awt.Color color)</code>	设置指定表格单元的前景颜色。参数 <code>x</code> 是要设置颜色的表格单元的坐标 <code>x</code> 值；参数 <code>y</code> 是要设置颜色的表格单元的坐标 <code>y</code> 值；参数 <code>color</code> 是要设置的颜色对象
<code>setBackground(int x,int y,java.awt.Color color)</code>	设置指定表格单元的背景颜色

说明：Table 类实现了 `com.javareport.ReportConstants` 接口，所有常量在 `ReportConstants` 接口中做了定义，其常量与值的对应情况如下：

- `Table.H_LEFT`：横向左对齐，值为 1；
- `Table.H_CENTER`：横向居中对齐，值为 2；
- `Table.H_RIGHT`：横向右对齐，值为 4；
- `Table.V_TOP`：竖向上对齐，值为 8；
- `Table.V_CENTER`：竖向居中对齐，值为 16；
- `Table.V_BOTTOM`：竖向下对齐，值为 32。

#### 4. Chart 类

Char 类即 `com.javareport.beans.Chart`，图表类。这个类的对象属于报表对象 `Report` 中的元素。统计图可以使用户浏览数据更加直观，开发人员可以使用这个类生成十几种报表统计图。

Chart 类的构造函数声明情况如下：

```
public Chart(java.lang.Number[][] data)
```

参数 `data` 是用于初始化图形中数据的二维数字型数据数组。

##### (1) `setLabel()`

```
public void setLabel(int i,java.lang.String label)
```

此方法用于设置图表中指定的单元数据的显示标签。参数 `i` 是需要设置标签的数据单元的下标值；参数 `label` 指定对应的单元数据的标签。

##### (2) `setLabels()`

```
public void setLabels(java.lang.String[] labels)
```

此方法设置图表中的单元数据显示的标签。参数 `labels` 是单元数据的标签字符串数组。

##### (3) `setData()`

此方法用于设置图表中指定的单元或所有单元的数据，有如下的 2 种形式：

```
public void setData(int i,int j,java.lang.Number data)
```

或：

```
public void setData(java.lang.Number[][] data)
```

第 1 种形式中的参数 `i` 指定二维数据单元的坐标 `x` 的值；参数 `j` 指定二维数据单元的坐标 `y` 的值；参数 `data` 用于给指定的单元赋值，数据可以是 `Byte`、`Double`、`Float`、`Integer`、`Long`、`Short`，这些数据类型都是 `Number` 类的子类。第 2 种形式中的参数 `data` 是一个二维的数据，数据可以是 `Byte`、`Double`、`Float`、`Integer`、`Long`、`Short`。



#### (4) setStyle()

```
public void setStyle(int type)
```

此方法设置统计图的类型，统计图可以是：曲线图、百分比图等。参数 **type** 是指定的统计图类型，总共有十多种，常用常量来表示，常用的有以下几种。

Chart.CHART\_PIE3D：立体饼图；

Chart.CHART\_STACKBAR3D：立体条形图；

Chart.CHART\_CURVE：曲线图；

Chart.CHART\_LINE：线图；

Chart.CHART\_POINT：点图；

Chart.CHART\_INVERTED\_CURVE：反向曲线图；

Chart.CHART\_INVERTED\_LINE：反向线图；

Chart.CHART\_INVERTED\_STACKBAR：横向的条形图。

#### (5) setShowValue()

```
public void setShowValue(boolean flag)
```

设置统计图中显示时是否把具体的数值也显示出来，参数 **flag** 是一个布尔型的标志值，**true** 表示显示，**false** 表示不显示。

### 5. RsTable 类

**RsTable** 类即 `com.javareport.beans.RsTable`，记录集表格类。这个类的对象属于报表对象 **Report** 中的元素。**RsTable** 类具有 **Table** 类的全部功能，是针对统计报表中显示记录集频繁使用的动作而设计的，开发者使用它时用几行代码就可把一个 **JDBC** 记录集里的数据以表格形式列举显示出来。

**RsTable** 类的构造函数用于实例化一个记录集表格对象，有如下的两种形式：

```
public RsTable(java.sql.ResultSet rs)
```

或

```
public RsTable(java.lang.String[] as, java.sql.ResultSet rs)
```

参数 **rs** 是填充表格内容的记录集；参数 **as** 是记录集中列的名称映射表。

## 11.3.3 如何开发 Web 图形与报表

### 1. 在 JSP 中开发 Web 图形与报表

自定义的 JSP 需要从 **WebReportEngine** 类继承下来：

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ page import="javax.servlet.*"%>
<%@ page import="com.javareport.beans.*"%>
<%@ page extends="com.javareport.http.WebReportEngine"%>
<%!
public Report createReport(HttpServletRequest request) throws Exception {
... ..
}
... ..
%>
```

这是一个开发的程序模板，一般情况下实现 `createReport()` 函数就可以，项目中报表的样式和内容就在这函数里实现，因此它是动态实时的报表，另外，还可以重载其他函数，可根据需要而定。

根据以上的代码，大多数情况下重载了 `createReport()` 方法就能满足需要了，报表的样式和内容等设置就在这函数里来实现；如果需要也可以重载其他函数。更详细的一个编程参考如下：

#### JSP 报表开发模板

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ page import="javax.servlet.*"%>
<%@ page import="com.javareport.beans.*"%>
<%@ page extends="com.javareport.http.WebReportEngine"%>
<%!
/*****
 * 这是报表系统在应用中给开发人员的 JSP 模板文件，可以根据需要调整接口内容。部分函
 * 数可以适当删除。在开发中一般是实现 createReport () 函数就可以，形成实时动态报表
 * 就是在这个函数里实现的。剩下的工作（怎样在 Web 上显示，怎样形成 Work, PDF 文件等）
 * 交给报表引擎自动实现。
 *****/
/**
 * 建立报表，返回报表的实例。这个报表实例可以在 Web 上显示，同时也可以导出 Word, Excel,
 * PDF, CSV, HTML 等格式的文档供使用。
 */
public Report createReport(HttpServletRequest request) throws Exception{
    Report report = new Report();
    report.addText("This is a template !");
    return report;
}
/**
 * 这是对上一个页面 Form 提交的参数进行检查，由于实时报表需要动态的参数，在这里进
 * 行数据校验。
 * 返回值为 null 时代表通过，其他内容则为参数错误的提示信息。
 */
public String validate(HttpServletRequest request){
    return null;
}
/**
 * 这是报表在 Web 上显示时，内容显示出来前执行的脚本，脚本内容一般为 JavaScript 脚
 * 本或 VBScript 脚本。
 * 返回值为 null 时代表通过没有脚本内容。
 */
public String getStartScript(HttpServletRequest request){
    return null;
}
/**
 * 这是报表在 Web 上显示时，内容显示出来后执行的脚本，脚本内容一般为 JavaScript 脚
 * 本或 VBScript 脚本。
 */
```



```

* 返回值为 null 时代表通过没有脚本内容。
*/
public String getEndScript(HttpServletRequest request){
    return null;
}
/**
* 这是报表在 Web 上显示时，上面的工具栏为标准的样式（上下翻页，导出文件）。可以在
* 此扩展工具栏的内容，一般可以添加公司主页的链接，返回上一层链接的“返回”按钮就是
* 在这里添加脚本的。
* 返回值为 null 时代表不添加内容。
*/
public String getToolbarScript(HttpServletRequest request){
    return null;
}
/**
* 这是报表在 Web 上显示时，如果不想让工具栏显示出来，就让函数的返回值就 false 就可以。
* 注意：如果是多页报表，上下翻页按钮就无法使用。
*/
public boolean isShowToolbar(){
    return true;
}
/**
* 这是报表在 Web 上显示时，导出文件的按钮可以自定义，比如应用中只要导出 PDF 文件，
* 其他的不需要，就可以这里设定。按钮值从 Work 按钮开始是（1，2，4，8，...），需要
* 显示的按钮则将它们值相加就可以了。
*/
public int getAllEchoButton(){
    return 0xFFFF;
}
%>

```

## 2. 在 Servlet 中开发 Web 报表

自定义的 Servlet 需要从 WebReportEngine 类继承下来，如下所示：

```

import javax.servlet.http.*;
import com.javareport.beans.*;
public class ReportExam extends WebReportEngine {
    public Report createReport(HttpServletRequest request) throws Exception {
        .....
    }
    .....
}

```

下面给出一个 Servlet 报表开发的详细模板，以供参考。

### Servlet 报表开发模板

```

<%@ page contentType="text/html; charset=GBK" %>
import javax.servlet.http.*;
import com.javareport.beans.*;
public class Template extends WebReportEngine {
    /*****
    * 这是报表系统在中应用中给开发人员的 JSP 模板文件，可以根据需要调整接口内容。部分函
    * 数可以适当删除。在开发中一般是实现 createReport () 函数就可以，形成实时动态报表
    */

```



```

    * 就是在这个函数里实现的。剩下的工作（怎样在 Web 上显示，怎样形成 Work，PDF 文件等）
    * 交给报表引擎自动实现。
    *****/
/**
    * 建立报表，返回报表的实例。这个报表实例可以在 Web 上显示，同时也可以导出 Word，Excel，
    * PDF，CSV，HTML 等格式的文档供使用。
    */
public Report createReport(HttpServletRequest request) throws Exception{
    Report report = new Report();
    report.addText("This is a template !");
    return report;
}
/**
    * 这是对上一个页面 Form 提交的参数进行检查，由于实时报表需要动态的参数，在这里进
    * 行数据校验。
    * 返回值为 null 时代表通过，其他内容则为参数错误的提示信息。
    */
public String validate(HttpServletRequest request){
    return null;
}
/**
    * 这是报表在 Web 上显示时，内容显示出来前执行的脚本，脚本内容一般为 JavaScript 脚
    * 本或 VBScript 脚本。
    * 返回值为 null 时代表通过没有脚本内容。
    */
public String getStartScript(HttpServletRequest request){
    return null;
}
/**
    * 这是报表在 Web 上显示时，内容显示出来后执行的脚本，脚本内容一般为 JavaScript 脚
    * 本或 VBScript 脚本。
    * 返回值为 null 时代表通过没有脚本内容。
    */
public String getEndScript(HttpServletRequest request){
    return null;
}
/**
    * 这是报表在 Web 上显示时，上面的工具栏为标准的样式（上下翻页，导出文件）。可以在
    * 此扩展工具栏的内容，一般可以添加公司主页的链接，返回上一层链接的“返回”按钮就是
    * 在这里添加脚本的。
    * 返回值为 null 时代表不添加内容。
    */
public String getToolbarScript(HttpServletRequest request){
    return null;
}
/**
    * 这是报表在 Web 上显示时，如果不想让工具栏显示出来，就让函数的返回值就 false 就可以。
    * 注意：如果是多页报表，上下翻页按钮就无法使用。
    */
public boolean isShowToolbar(){
    return true;
}
/**

```



- \* 这是报表在 Web 上显示时，导出文件的按钮可以自定义，比如应用中只要导出 PDF 文件，
- \* 其他的不需要，就可以这里设定。按钮值从 Work 按钮开始是 (1, 2, 4, 8, ...)，需要
- \* 显示的按钮则将它们值相加就可以了。

```
*/
public int getAllEchoButton(){
    return 0xFFFF;
}
%>
```

### 11.3.4 开发统计图

JavaReport 只需要少量的代码就能输出美观而又实用的 Web 图形，来看实例。

#### 【实例 11-7】开发 Web 统计图

webChart.jsp

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ page import="com.javareport.beans.*"%>
<%@ page extends="com.javareport.http.WebReportEngine"%>
<%!
public Report createReport(HttpServletRequest request) throws Exception{
    //图片类型数组
    int[] chartType = new int[]{
        Chart.CHART_PIE3D,Chart.CHART_STACKBAR3D,
        Chart.CHART_CURVE,Chart.CHART_LINE,
        Chart.CHART_POINT,Chart.CHART_INVERTED_CURVE,
        Chart.CHART_INVERTED_LINE,Chart.CHART_INVERTED_STACKBAR};
    //单元数据的显示标签字符串数组
    String[] labels = new String[] {"长沙市","衡阳市","怀化市","岳阳市"};
    //实例化报表对象
    Report report = new Report();
    //在页眉中添加文本信息内容
    report.addHeaderText("湖南省 2011 年继续教育报名人数统计图");
    //在报表的页眉添加一条横直线
    report.addHeaderSeparator(1);
    //在页尾添加一条横直线
    report.addFooterSeparator(1);
    //在页尾添加文本信息内容
    report.addFooterText("第{P}页， 共{N}页");
    //循环输出各种类型的图片
    for (int i = 0; i < chartType.length; i++) {
        try {
            //实例化一个图表对象
            Chart chart = new Chart((Number[][] )getData(request));
            //设置图表中的单元数据的显示的标签
            chart.setLabels(labels);
            //设置统计图的类型
            chart.setStyle(chartType[i]);
            //设置统计图中显示的时候把具体的数值也显示出来
            chart.setShowValue(true);
            //在报表中添加文本信息内容
            report.addText("报表中常见的报表统计图表("+i+"): ");
```

```

        //在报表中添加图表信息内容
        report.addChart(chart);
        //在报表中添加换行符号
        report.addBreak();
        report.addBreak();
        report.addBreak();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
return report;
}
//读者可根据需要设置数组的值, 或从数据库中取出值放入数组中以动态显示数据
public Double[][] getData(HttpServletRequest request){
    Double[][] data = new Double[1][4];
    data[0][0] = new Double(50);
    data[0][1] = new Double(50);
    data[0][2] = new Double(35);
    data[0][3] = new Double(55);
    return data;
}
//定制Web 报表在页面首部显示的工栏为标准的样式, 增加一个"返回"按钮, 返回到首页
public String getToolbarScript(HttpServletRequest request){
    return "<a href=\"webChart.jsp\"><img src=\"\"+request.getRequestURI()+
        \"?op=Resource&name=/resource/back.gif\" border=\"0\" alt=\"返回\"></a>";
}
%>

```

程序运行结果如图 11-9 所示。

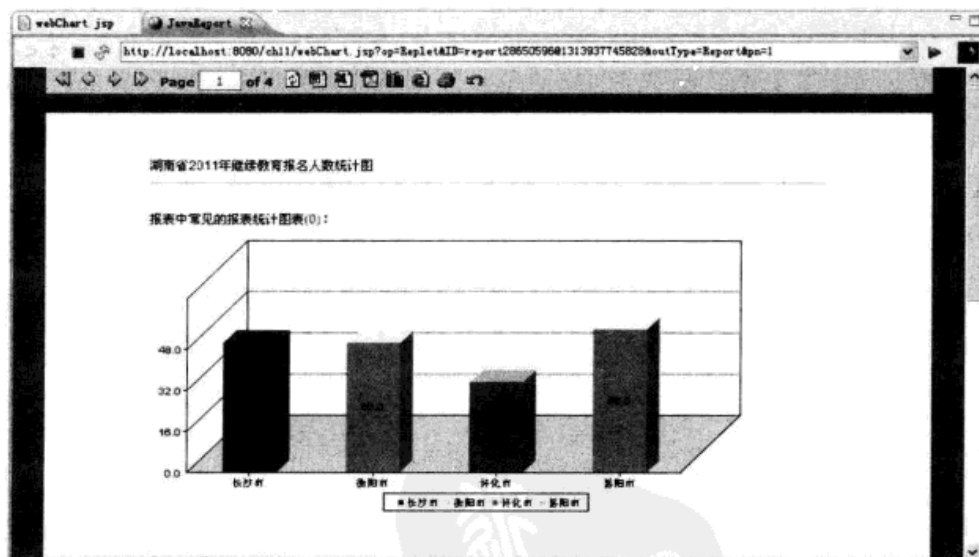


图 11-9 Web 统计图

程序中首先重载了 `createReport()` 方法, 在这个方法的方法体中, 先用 “`new Report()`” 生成了一个报表 `Report` 对象 `report`, 再设置了报表的页眉页脚, 然后再用一个 `for` 循环输出了 8 种 Web 统计图。



重载 `getData()` 方法的目的是为了设置显示的 Web 统计图所要表示的数据, 这个方法的返回值是一个 `Double` 类型的二维数组, 由于第一维只有一行, 即相当于一维数据。方法的返回值就是 Web 统计图要表示的数据。实际工程中一维数据的数据来源常常来自于对数据库中数据查询的结果。

重载 `getToolBarScript()` 方法的目的是为了定制 Web 报表在页面首部显示的工具栏为标准样式, 增加了一个“返回”按钮, 当单击时返回到网站的首页。“返回”按钮的图形在 `JavaReport` 组件包的 `resource` 文件夹中, 读者可以将 `JavaReport` 组件包的 `rar` 文件解压缩即可看到。



**提示** 从运行结果图来看, 访问地址是比较冗长的, 在访问这个页面时不必带这么多参数, 直接用 `http://localhost:8080/ch11/webChart.jsp` 即可, 图中的地址是访问后系统自动生成的。

本例中的代码只是实现了一维数据的展现, 有时也需要展现二维的数据。二维数据图形与一维数据图形的区别就是在数据展现上加大了数据的展现量, 在同一个单元数据标签处可显示属于同一个单元数据的多个数据。下面修改一下本例中 `getData()` 方法的内容, 修改后的代码如下:

```
public Double[][] getData(HttpServletRequest request){
    Double[][] data = new Double[4][4];
    data[0][0] = new Double(200); data[0][1] = new Double(250);
    data[0][2] = new Double(220); data[0][3] = new Double(280);
    data[1][0] = new Double(500); data[1][1] = new Double(700);
    data[1][2] = new Double(520); data[1][3] = new Double(900);
    data[2][0] = new Double(350); data[2][1] = new Double(400);
    data[2][2] = new Double(380); data[2][3] = new Double(320);
    data[3][0] = new Double(550); data[3][1] = new Double(590);
    data[3][2] = new Double(337); data[3][3] = new Double(340);
    return data;
}
```

则程序运行的结果如图 11-10 所示。

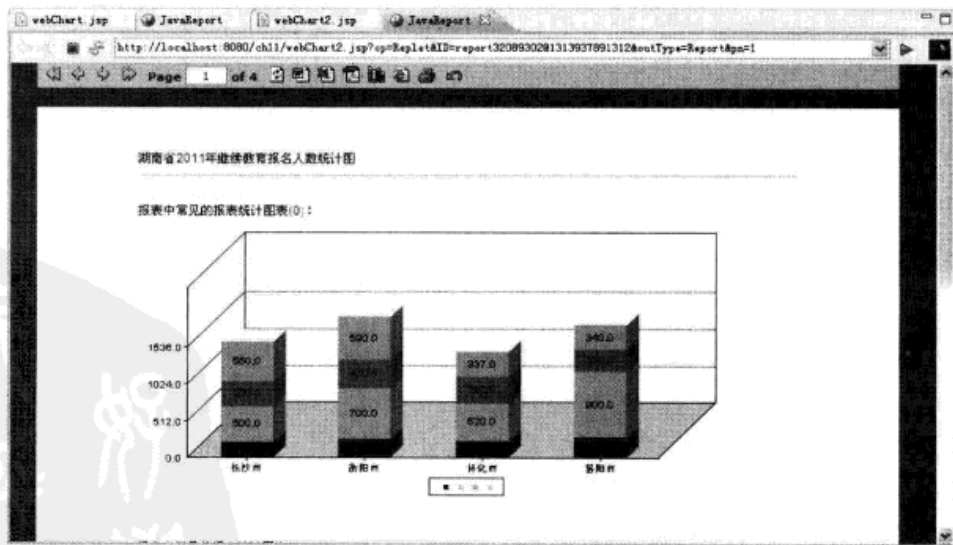


图 11-10 二维统计图

## 11.3.5 开发 Web 报表

### 【实例 11-8】开发 Web 统计报表

table.jsp

```

<%@ page contentType="text/html; charset=GBK" %>
<%@ page import="java.awt.*"%>
<%@ page import="com.javareport.beans.*"%>
<%@ page extends="com.javareport.http.WebReportEngine"%>
<%!
public Report createReport(HttpServletRequest request) throws Exception{
    //实例化报表对象
    Report report = new Report();
    //在页眉中添加文本信息内容
    report.addHeaderText("湖南物流集团 2011 年新增客户数分区统计表");
    //在报表的页眉添加一条横直线
    report.addHeaderSeparator(1);
    //在页尾添加一条横直线
    report.addFooterSeparator(1);
    //在页尾添加文本信息内容
    report.addFooterText("第{P}页, 共{N}页");

    //在报表中添加文本信息内容
    report.addText("销售情况一览表: ");
    //在报表中添加换行符号
    report.addBreak();
    //在报表中添加表格
    report.addTable(getTableA());
    //在报表中添加换行符号
    report.addBreak();

    //在报表中添加文本信息内容
    report.addText("销售情况一览表(合并表格): ");
    //在报表中添加换行符号
    report.addBreak();
    //在报表中添加表格
    report.addTable(getTable());
    //在报表中添加换行符号
    report.addBreak();

    return report;
}
//-----得到销售情况一览表对象-----
public Table getTableA(){
    String[][] data = getData();
    Table table = new Table(data);
    table.setColBorder(0);
    table.setRowBorder(0);
    table.setRowBorder(0,1);
    table.setRowBackground(0,new Color(128,0,0));
    table.setRowForeground(0,Color.white);
}

```



```

        table.setRowBackground(1,new Color(255,255,128));
        table.setRowForeground(1,Color.black);
        table.setRowBackground(2,new Color(255,255,128));
        table.setRowForeground(2,Color.black);
        table.setRowBackground(3,new Color(255,255,128));
        table.setRowForeground(3,Color.black);
        table.setRowBackground(4,new Color(255,255,128));
        table.setRowForeground(4,Color.black);
        table.setRowBackground(5,new Color(255,255,128));
        table.setRowForeground(5,Color.black);
        return table;
    }
    //-----得到销售情况一览表(合并表格)对象-----
    public Table getTable(){
        String[][] data = getTotalData();
        Table table = new Table(data);
        table.setAlignment(Table.H_CENTER + Table.V_CENTER);
        table.setColAutoSize(true);
        table.setRowBackground(0,Color.LIGHT_GRAY);
        table.setRowBackground(1,Color.LIGHT_GRAY);
        table.setColBackground(0,Color.LIGHT_GRAY);
        table.setRowBackground(7,new Color(255,255,128));
        table.setHeaderRowCount(2);
        table.setHeaderColCount(1);
        table.setRowBorder(table.LINE_THIN);
        table.setColBorder(table.LINE_THIN);
        table.setCellSpan(0,0,new Dimension(1,2));
        table.setCellSpan(0,1,new Dimension(2,1));
        table.setCellSpan(0,3,new Dimension(2,1));
        table.setCellSpan(0,3,new Dimension(2,1));
        return table;
    }
    //生成销售情况数据,实际工程中一般从数据库中获取
    public String[][] getData(){
        String[][] data = new String[6][4];
        data[0][0]="区域"; data[0][1]="第一季度"; data[0][2] = "第二季度";
        data[0][3]="第三季度";
        data[1][0]="华南地区";data[1][1]="¥2,000";data[1][2]="¥2,500";
        data[1][3]="¥2,200";
        data[2][0]="华东地区";data[2][1]="¥6,000";data[2][2]="¥4,500";
        data[2][3]="¥4,800";
        data[3][0]="华中地区";data[3][1]="¥500";data[3][2]="¥400";
        data[3][3]="¥700";
        data[4][0]="华北地区";data[4][1]="¥3,000";data[4][2]="¥3,200";
        data[4][3]="¥2,500";
        data[5][0]="东北地区";data[5][1]="¥4,000";data[5][2]="¥5,000";
        data[5][3]="¥4,400";
        return data;
    }
    //得到销售汇总统计数据,实际工程中一般从数据库中获取
    public String[][] getTotalData(){
        String[][] data = new String[8][4];
        data[0][0]="区域";data[0][1]="上半年";data[0][3]="下半年";

```



```

data[1][1]="第一季度";data[1][2]="第二季度";data[1][3]="第三季度";
data[2][0]="华南地区";data[2][1]="¥2,000";data[2][2]="¥2,500";
data[2][3]="¥2,200";
data[3][0]="华东地区";data[3][1]="¥6,000";data[3][2]="¥4,500";
data[3][3]="¥4,800";
data[4][0]="华中地区";data[4][1]="¥500";data[4][2]="¥400";
data[4][3]="¥700";
data[5][0]="华北地区";data[5][1]="¥3,000";data[5][2]="¥3,200";
data[5][3]="¥2,500";
data[6][0]="东北地区";data[6][1]="¥4,000";data[6][2]="¥5,000";
data[6][3]="¥4,400";
data[7][0]="总计";data[7][1]="¥15,500";data[7][2]="¥15,600";
data[7][3]="¥14,600";
return data;
}
%>

```

程序运行结果如图 11-11 所示。

湖南物流集团2011年新增客户数分区统计表

销售情况一览表:

区域	第一季度	第二季度	第三季度
华南地区	¥ 2,000	¥ 2,500	¥ 2,200
华东地区	¥ 6,000	¥ 4,500	¥ 4,800
华中地区	¥ 500	¥ 400	¥ 700
华北地区	¥ 3,000	¥ 3,200	¥ 2,500
东北地区	¥ 4,000	¥ 5,000	¥ 4,400

销售情况一览表(合并表格):

区域	上半年		下半年
	第一季度	第二季度	第三季度
华南地区	¥ 2,000	¥ 2,500	¥ 2,200
华东地区	¥ 6,000	¥ 4,500	¥ 4,800
华中地区	¥ 500	¥ 400	¥ 700
华北地区	¥ 3,000	¥ 3,200	¥ 2,500
东北地区	¥ 4,000	¥ 5,000	¥ 4,400
总计	¥ 15,500	¥ 15,600	¥ 14,600

图 11-11 Web 统计报表

读者对照图 11-11 再分析一下源代码应当不难。程序中用到了 `table` 类的许多方法，用于设置表格中的各种格式，如合并单元格、背景颜色、前景颜色等。

`report.addTable()` 方法的参数是一个 `Table` 对象，为生成图 11-11 所示的两个报表，分别用了方法 `getTableA()` 和 `getTable()` 来生成 `Table` 对象。生成 `Table` 对象时需要填充表格中的数据，以及设置表格显示时的背景颜色、前景颜色等格式。

如何构造出需要合并单元格时的合并格式呢？在设置 `Table` 数据数组中的数据时，只需要不设置相应的单元格的数据值即可。比如第 2 个表格共行 8 行 4 列，也即一个 8 行 4 列的数据数组，单元格“区域”合并了两个竖向的单元格，则只需要不设置数组的第 2 行第 1 列的值（即 `data[1][0]`，因为数组的下标是从 0 开始的）即可；另有一个单元格“上半年”合并了两个横向的单元格，则只需要不设置数组的第 1 行第 3 列的值（即 `data[0][2]`）即可。





**提示** 实际工程中，报表的数据往往来自于数据库中，因此需要在 `getTotalData()` 方法中编写代码从数据库中得到统计数据，统计的方法是可以使用特定的 SQL 语句或取出数据后在 Java 语句中统计。

输出报表还有一种更直接的方法，就是用 SQL 语句查询出数据库中的数据后会得到一个 `ResultSet` 对象，如 `rs`，再用如下的语句：

```
RsTable rsTable = new RsTable(rs)
```

`rsTable` 对象取为 `RsTable` 类的一个实例。重载如下的方法：

```
public RsTable getRsTable(HttpServletRequest request) throws Exception
```

将返回值设为 `rsTable` 即可将数据库查询的结果作为二维表格中的数据直接输出，而不必再转换成二维数组中的数据再输出到报表中，这样可以大大减少程序员需要编写的代码量。

## 11.4 生成验证码

读者可能经常在一些论坛发表帖子或用户登录操作时会遇到要求用户输入验证码的情况，网站会在验证码输入框后生成一张验证码的图片，只有验证码输入正确后才能继续下一步操作。

在论坛中发表帖子时，一般的 Web 程序都是以提交表单形式来提交帖子的内容，接收数据后插入到数据库的表中；如果有人不怀好意，利用一些自动提交表单的工具来不断地提交表单，则会导致数据库中的数据迅速膨胀，很快就会导致数据库的空间被用光。如果使用了验证码，由于验证码生成的是图片，程序很难识别出来，而且图片中还可以加入干扰，即便是使用了 OCR（影像识别）技术也很难识别出来，当验证码没有通过时，数据就不会被提交到数据库中（需要开发人员在程序中进行控制）。

### 【实例 11-9】生成彩色验证码

本实例为用户登录页面生成了彩色验证码，这样用户在登录时就要输入用户名、密码和验证码，只有这三个输入元素同时通过程序的验证才能进入系统。其他情况下需要用到验证码时，读者可以参照这个实例来进行制作。

先来把生成验证码图片的程序封装为一个 `JavaBean`，这个 `JavaBean` 的源代码如下。

#### MakeCertPic.java

```
package cert;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.OutputStream;
import java.util.Random;
import javax.imageio.ImageIO;
/**
 * @author dengziyun
```



```

* 生成验证码图片
*/
public class MakeCertPic {
    //验证码图片中可以出现的字符集，可根据需要修改
    private char mapTable[]={
        'a','b','c','d','e','f',
        'g','h','i','j','k',
        'm','n','p','q','r',
        's','t','u','v','w','x',
        'y','z','0','2','3',
        '4','5','6','7','8','9'};

    /**
     * 功能:生成彩色验证码图片
     * 参数 width 为生成图片的宽度,, 参数 height 为生成图片的高度
     * 参数 os 为页面的输出流
     */
    public String getCertPic(int width, int height, OutputStream os) {
        if(width<=0)width=60;
        if(height<=0)height=20;
        BufferedImage image = new BufferedImage(width, height,
            BufferedImage.TYPE_INT_RGB);
        // 获取图形上下文
        Graphics g = image.getGraphics();
        // 设定背景色
        g.setColor(new Color(0xDCDCDC));
        g.fillRect(0, 0, width, height);
        //画边框
        g.setColor(Color.black);
        g.drawRect(0,0,width-1,height-1);
        // 取随机产生的认证码
        String strEnsure = "";
        // 4 代表 4 位验证码,如果要生成更多位的认证码,则加大数值
        for(int i=0; i<4; ++i) {
            strEnsure += mapTable[(int)(mapTable.length*Math.random())];
        }
        // 将认证码显示到图像中, 如果要生成更多位的认证码, 增加 drawString 语句
        g.setColor(Color.black);
        g.setFont(new Font("Atlantic Inline",Font.PLAIN,18));
        String str = strEnsure.substring(0,1);
        g.drawString(str,8,17);
        str = strEnsure.substring(1,2);
        g.drawString(str,20,15);
        str = strEnsure.substring(2,3);
        g.drawString(str,35,18);
        str = strEnsure.substring(3,4);
        g.drawString(str,45,15);
        // 随机产生 10 个干扰点
        Random rand = new Random();
        for (int i=0;i<10;i++) {
            int x = rand.nextInt(width);
            int y = rand.nextInt(height);
            g.drawOval(x,y,1,1);
        }
    }
}

```



```

        // 释放图形上下文
        g.dispose();
        try {
            // 输出图像到页面
            ImageIO.write(image, "JPEG", os);
        } catch (IOException e) {
            return "";
        }
        return strEnsure;
    }
}

```

在 `getCertPic()` 方法中, 首先创建了一个内存图像的实例对象, 再得到此内存图像的图形上下文对象, 接着再用这个上下文对象画背景、边框。接下来, 随机生成 4 个在 `mapTable[]` 数组中的字符, 组成字符串作为验证字符串, 并输出在内存中, 为了造成一定的干扰, 随机画了 10 个干扰点, 如果要加大干扰效果, 可再多画一些点, 也就是说可以加大 `for` 循环的结束值。



**提示** 在生成验证码的列表 (本例中为 `mapTable` 数组) 中, 应尽量回避容易产生混淆的字符, 如 “i” (字母)、 “l” (字母) 和 “1” (数字), “o” (字母) 和 “0” 数字。

再编写一个 `Servlet` 来显示生成的验证码, 有了 `Servlet` 后, 即可以在登录的 `JSP` 页面中使用这个 `Servlet`。`Servlet` 的源代码如下。

#### ShowCertPic.java

```

package cert;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ShowCertPic extends HttpServlet{
    public void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        MakeCertPic image=new MakeCertPic();
        String str = image.getCertPic(0,0,response.getOutputStream());
        //将验证码存入 session
        request.getSession().setAttribute("certCode2", str);
    }
}

```

这个 `Servlet` 中的代码比较简单, `service()` 方法中只有 3 行代码, 功能是生成图片并输出, 再将生成的图片代表的 4 个字符存入一个 `session` 变量 `certCode2` 中。

`Servlet` 开发完成后需要在当前 `Web` 应用的 `web.xml` 文件中做出部署, 本例中部署的配置如下:

```

<servlet>
    <servlet-name>ShowCertPic</servlet-name>
    <servlet-class>cert.ShowCertPic</servlet-class>
</servlet>
<servlet-mapping>

```

```
<servlet-name>ShowCertPic</servlet-name>
<url-pattern>/ShowCertPic</url-pattern>
</servlet-mapping>
```

接下来再看登录的 JSP 页面源代码。

## LoginPic.jsp

[illegible]

登录页面的运行结果如图 11-12 所示。



图 11-12 生成登录页面中的验证码

生成验证码的语句如下所示:

```

```

ShowCertPic 是在 web.xml 文件中定义的 Servlet 名字。

验证码的输入是否正确可用如下语句验证:

```
String certCode=request.getParameter("certCode");
if(certCode.equals((String)session.getAttribute("certCode2"))){
    out.print("验证码输入正确");
}
else{
    out.print("验证码输入错误");
}
```



## 11.5 发送和接收邮件

JavaMail 对 SMTP、POP3、IMAP 提供了支持，封装了电子邮件处理中用到的邮件对象、发送功能、身份认证功能及接收邮件功能等。

### 11.5.1 下载与安装 JavaMail

如果使用的 JDK 为 JDK 7，即本书中使用的开发环境，则只需下载 JavaMail 组件包即可。JavaMail 组件包可以从 Sun 公司的网站上免费下载得到，JavaMail 的官方网址如下：

<http://www.oracle.com/technetwork/java/javamail/index.html>

通过如下的网页可以得到 JavaMail 的下载地址，从而得到这个组件包：

<http://www.oracle.com/technetwork/java/javamail/index-138643.html>

下载得到的是一个 ZIP 压缩包。至本书成稿之日，JavaMail 组件包的最新版本为 1.4.4，下载得到的文件为 javamail1\_4\_4.zip。解压缩后，可以得到 JavaMail 组件的 API 文档及相关的 JAR 文件，其中 mail.jar 是进行邮件开发所必需的。

将 mail.jar 复制到 Tomcat 安装目录的 lib 子目录中则所有 Web 应用均可用；如果只需要当前 Web 应用可用则可复制到当前 Web 应用的“WEB-INF/lib”目录中，如果没有此目录，则可以创建一个。

如果您使用的是 JDK 6 以下的版本，则还需要安装 JAF（JavaBeans Activation Framework）以提供 javax.activation 包。在 JDK 6 以上版本中已包含这个包（在 rt.jar 中），故不必再进行安装了。



**提示** 随书光盘中带有 mail.jar，位于本章的 Web 应用的“WEB-INF/lib”目录中。

### 11.5.2 JavaMail 常用的 API

要使用 JavaMail 组件来收发邮件，当然需要了解 JavaMail 组件的 API 了，如果需要深入应用则更需要详细理解。



**提示** 读者在阅读时也可以跳过本节 API 的说明而直接进入实例的学习，在程序中使用到了 JavaMail API 时再回到本节来查看相关属性、方法的说明。

如图 11-13 所示，图中显示了 JavaMail 组件包中主要的接口和类，以及它们之间的关系。通过 JavaMail 组件包，开发人员主要可以完成如下的客户端开发功能：

- （1）创建一封邮件。JavaMail 组件使用 Part 接口和 Message 类来定义一封邮件，使用 JAF 中的 DataHandler 来将邮件打包处理。
- （2）创建一个 Session 对象，以进行用户鉴别，控制对邮箱的存取、发送操作。

- (3) 向一个收件箱列表中的邮箱发送邮件。
- (4) 从某个邮箱中收取邮件。

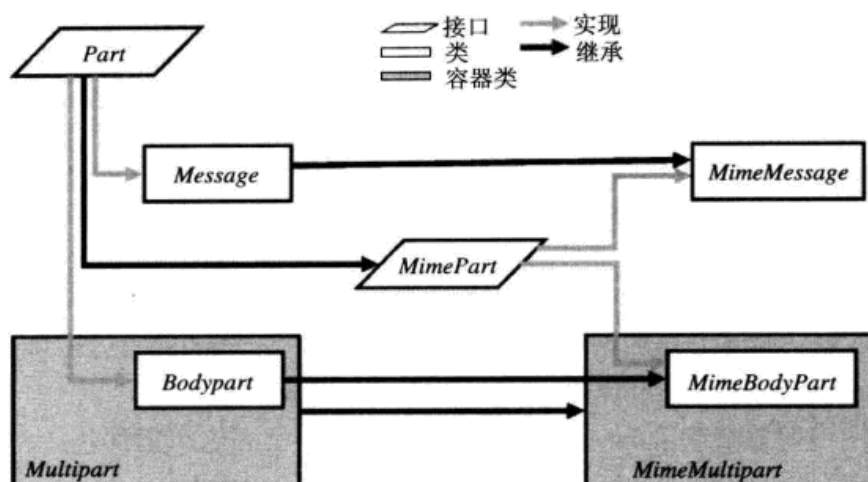


图 11-13 JavaMail 组件包中主要的接口和类

!

**提示** JavaMail 中的 Session 指的是邮件会话，而非 JSP 的内置对象 Session。

以发送邮件处理过程为例，如图 11-14 所示。**Message** 类封装邮件，再通过 **Transport** 类来发送邮件，接收方邮箱收到邮件后，如果要收取邮件，则要使用到 **Store** 类、**Folders** 类、**Message** 类。**Store** 类代表存放邮件的信箱，**Folder** 类表示将邮件归类整理好了的文件夹。

使用 **JavaMail** 组件收到邮件需要熟悉这些类：**Session**、**InternetAddress**、**MimeMessage**、**Transport**、**Store**、**Folder** 等。

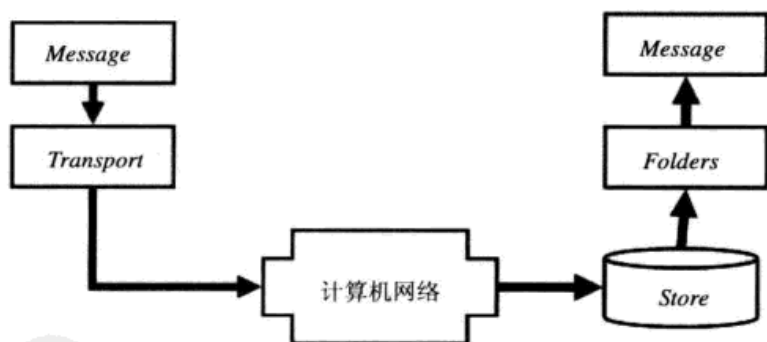


图 11-14 发送邮件的处理过程

## 1. Session 类

用户要收发邮件必须先建立邮件会话 **Session** 类的对象，这个对象可用来创建邮件对象、实现邮件对象中数据的封装，还可指定邮件服务器认证的客户端属性。

### (1) getInstance()

这个方法创建与具体客户端相关的 Session 对象，有以下两种形式：

```
public static Session getDefaultInstance(java.util.Properties props)
```



或

```
public static Session getDefaultInstance(java.util.Properties props,
    Authenticator authenticator)
```

参数 `props` 是属性对象，继承自 `HashTable`（哈希表），简单地说就是键值对，参数 `authenticator` 对象是邮件认证对象。

### (2) getInstance()

此方法的功能是创建会话对象，调用形式与 `getDefaultInstance()` 方法类似，有以下两种声明情况：

```
public static Session getInstance(java.util.Properties props)
```

或

```
public static Session getInstance(java.util.Properties props,
    Authenticator authenticator)
```

### (3) getProperties()

此方法得到邮件会话的属性类对象，声明情况如下：

```
public java.util.Properties getProperties()
```

如果要直接得到具体的某个属性的值则使用如下的方法：

```
public java.lang.String getProperty(java.lang.String name)
```

### (4) getStore()

得到邮件会话的邮件存储 `Store` 对象，`Store` 对象可以建立与邮件服务器的连接，可以从邮件服务器中得到邮件。

```
public Store getStore(java.lang.String protocol) throws NoSuchProviderException
```

参数 `protocol` 是使用的协议，要从邮件服务中获取邮件就要使用 `POP3` 或 `IMAP` 协议。如果得不到 `Store` 对象则会抛出 `NoSuchProviderException` 异常。

### (5) getTransport()

得到一个可以传送消息到特定地址的传送对象。方法的形式有：

```
public Transport getTransport(Address address) throws NoSuchProviderException
```

或

```
public Transport getTransport(java.lang.String protocol)
    throws NoSuchProviderException
```

参数 `address` 是要传送至的目的地址对象，参数 `protocol` 是使用的协议。如果得不到传送对象则会抛出 `NoSuchProviderException` 异常。

## 2. InternetAddress 类

`InternetAddress` 代表邮件地址对象，声明情况如下：

```
public class InternetAddress extends Address implements java.lang.Cloneable
```

可见它继承自抽象类 `javax.mail.Address`。

### (1) 构造函数

构造函数用来建立 `InternetAddress` 类的对象，调用方法如下：

```
InternetAddress(String address)
```

其中，参数 `address` 是构造对象的 E-mail 地址，如 `dzycsai@csai.cn`，`dzycsai` 是笔者在 `csai.cn` 的邮箱用户名，`@` 是邮件的分隔符，`csai.cn` 是邮件服务器。

### (2) getAddress()

此方法得到 E-mail 地址，声明情况如下：

```
public java.lang.String getAddress()
```

## 3. MimeMessage 类

`MimeMessage` 代表 MIME 风格的邮件，它实现了 `Message` 抽象类和 `MimePart` 接口，声明情况如下：

```
public class MimeMessage extends Message implements MimePart
```

客户端如果想要创建一个 MIME 风格的邮件，就需要先生成一个空的 `MimeMessage` 对象，再填充这个对象的相关属性与邮件的内容。

`MimeMessage` 类有一个内部类 `Recipient` 类，即 `javax.mail.internet.MimeMessage.Recipient`，表示邮件消息的接收者，这个类有一个属性 `to` 表示接收者，即接收邮件的邮箱地址。

### (1) 构造函数

构造函数可生成邮件消息类对象，调用方法如下：

```
MimeMessage(Session session)
```

参数 `session` 是邮件会话对象，是用来建立邮件消息类对象的邮件会话对象。

### (2) addRecipients()

此方法用于增加一个接收邮件的邮箱对象。声明情况如下：

```
public void addRecipients(Message.RecipientType type, Address[] addresses)
    throws MessagingException
```

或

```
public void addRecipients(Message.RecipientType type, java.lang.String addresses)
    throws MessagingException
```

第 1 种形式用于增加多个邮件接收者；第 2 种情况只有 1 个邮件接收者。参数 `type` 是邮件接收类型，`addresses` 是增加的邮件发送地址。

### (3) getAllRecipients()

得到所有的邮件接收者。声明情形如下：

```
public Address[] getAllRecipients() throws MessagingException
```

与此方法相对应的设置邮件接收者的方法为：

```
public void setRecipients(Message.RecipientType type,
    java.lang.String addresses)
    throws MessagingException
```

如果 `address` 参数的值为 `null`，则表示无接收者。



#### (4) getContent()

得到邮件消息内容对象。声明情形如下：

```
public java.lang.Object getContent() throws java.io.IOException,
    MessagingException
```

与此方法对应的设置消息内容的方法为：

```
public void setContent(Multipart mp) throws MessagingException
```

#### (5) getContentType()

得到表示邮件消息内容类型的字符串，如"text/plain"。声明情形如下：

```
public java.lang.String getContentType() throws MessagingException
```

#### (6) getFrom()

得到邮件消息发件邮箱数组，返回类型为 Address 数组。声明情形如下：

```
public Address[] getFrom() throws MessagingException
```

发件邮箱位于邮件头中的 From 属性中，如果 From 中没有值则取出 Sender 属性中的值，如果 Sender 属性中也没有值则返回 null。

与此方法对应的设置发件邮箱的方法为：

```
public void setFrom() throws MessagingException
```

此方法将邮件头中的 From 属性值设为 InternetAddress.getLocalAddress()方法的返回值。

#### (7) getSubject()

得到邮件消息的主题，其实也就是邮件头中的 Subject 属性的值。声明情形如下：

```
public java.lang.String getSubject() throws MessagingException
```

与此方法对应的设置消息的主题的方法有：

```
public void setSubject(java.lang.String subject) throws MessagingException
public void setSubject(java.lang.String subject, java.lang.String charset)
    throws MessagingException
```

#### (8) isMimeType()

判断邮件消息内容是否是指定的类型，返回类型为 boolean，是则返回 true，否则返回 false。如 isMimeType("text/plain")。声明情形如下：

```
public boolean isMimeType(java.lang.String mimeType) throws MessagingException
```

### 4. Transport 类

Transport 是邮件发送类，类的层次结构为 javax.mail.Transport，可由 Session 类对象的 getTransport()方法生成。

#### (1) connect()

此方法用于连接到指定的邮件服务器。方法的声明情况如下：

```
public void connect(java.lang.String host, java.lang.String user,
    java.lang.String password) throws MessagingException
```

或

```
public void connect(java.lang.String user, java.lang.String password)
    throws MessagingException
```

或

```
public void connect(java.lang.String host,int port,java.lang.String user,
    java.lang.String password) throws MessagingException
```

参数 **SMTPserverName** 是用来发送邮件的 **SMTP** 服务器的地址, 如 **csai** 邮件服务器为 **smtp.csai.cn**; **UserName** 为登录邮件服务器的用户名; **Password** 是登录邮件服务器的用户密码; 如果是连接本地邮件服务器则不必给出服务器地址而采用第 2 种形式即可; 如果还有特定的端口号则使用第 3 种形式, 参数 **port** 为指定的端口号。

## (2) sendMessage()

此方法用于发送消息, 声明情况如下:

```
public abstract void sendMessage(Message msg,Address[] addresses)
    throws MessagingException
```

参数 **Message** 是要发送的邮件消息, 参数 **address** 是发往邮件地址的数组。

## 5. Store 类

**Store** 类可用来从邮件服务器上接收邮件, 其类层次结构为 **javax.mail.Store**。**Store** 类最常用的方法就是 **getFolder()**, 此方法得到指定名称的邮件服务器文件夹中的邮件, 调用方法如下:

```
StoreObjectName.getFolder(String name)
```

**StoreObjectName** 是 **Store** 类对象的名称; 参数 **name** 是指定的邮件服务器上文件夹的名称, 如 **INDEX**。

## 6. Folder 类

**Folder** 类是邮件文件夹类, 类的层次结构为 **javax.mail.Folder**。**Folder** 类有两个常见的属性, **READ\_ONLY** 表示只读, **READ\_WRITE** 表示其内容可读可写。

### (1) exists()

此方法验证邮件文件夹是否存在, 返回值是 **boolean** 类型, 如果存在则返回 **true**, 否则返回 **false**。

```
public abstract boolean exists() throws MessagingException
```

### (2) fetch()

根据获取邮件的选项从邮件服务器中获取邮件至 **Message** 数组中, 方法的形式如下:

```
public void fetch(Message[] msgs,FetchProfile fp) throws MessagingException
```

参数 **fp** 是取邮件的配置。**FetchProfile** 类的层次结构是 **javax.mail.FetchProfile**, 调用其 **add(FetchProfile.Item item)** 可加入配置, 常用的配置项有 **ENVELOP**; **FLAGS**; **Content\_INFO**。

### (3) getMessages()

得到此邮件文件夹对象的所有邮件消息对象数组, 返回类型是一个邮件消息数组。方法的形式如下:

```
public Message[] getMessages() throws MessagingException
```



### 11.5.3 发送邮件

#### 【实例 11-10】发送邮件

下面的程序将利用 126 的一个 Internet 邮箱作为发件箱，向 csai 的一个 Internet 邮箱发送一封简单的文本格式的邮件。

子包括两个 JSP 页面，第一个 JSP 页面 `writeMail.jsp` 用于输入要发送的邮件的邮件标题、邮件内容以及邮件接收者；另一个 JSP 页面 `sendMail.jsp` 用于接收输入的数据并发送邮件。

`writeMail.jsp`

```
<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
<title>写邮件内容</title>
</head>
<body>
<form name="form1" method="post" action="sendMail.jsp">
<table width="100" border="1" align="center" cellspacing="1">
  <tr>
    <td width="34%">收信人地址:</td>
    <td width="66%"><input name="to" type="text" id="to"></td>
  </tr>
  <tr>
    <td>主题:</td>
    <td><input name="title" type="text" id="title"></td>
  </tr>
  <tr>
    <td height="107" colspan="2">
      <textarea name="content" cols="50" rows="5" id="content"></textarea>
    </td>
  </tr>
  <tr align="center">
    <td colspan="2">
      <input type="submit" name="Submit" value="发送">
      <input type="reset" name="Submit2" value="重输">
    </td>
  </tr>
</table>
</form>
</body>
</html>
```

程序运行结果如图 11-15 所示。

有这个界面中输入接收邮件的邮箱、邮件的主题以及邮件的正文内容，单击“发送”按钮，即将数据提交到 `sendMail.jsp` 页面。

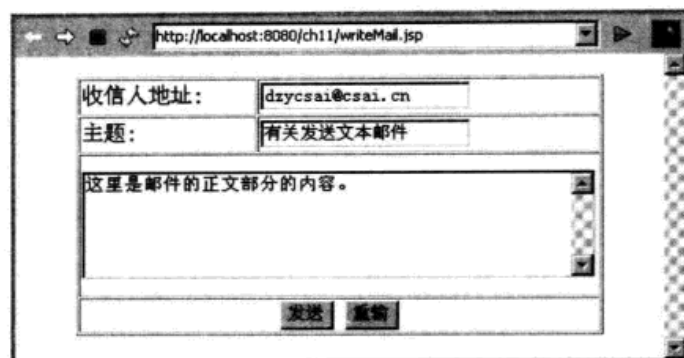


图 11-15 输入邮件内容

## sendMail.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ page import="java.util.*,javax.mail.*"%>
<%@ page import="javax.mail.internet.*"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>发送邮件的结果</title>
</head>
<body>
<fmt:requestEncoding value="gb2312"/>
<%
try{
//从html表单中接收邮件信息
String to_mail=request.getParameter("to");
String to_title=request.getParameter("title");
String to_content=request.getParameter("content");
//建立邮件会话
Properties props=new Properties();
props.put("mail.smtp.host","smtp.126.com");//存储发送邮件服务器的信息
props.put("mail.smtp.auth","true");//同时通过验证
Session s=Session.getInstance(props);//根据属性新建一个邮件会话
s.setDebug(true);
//由邮件会话新建一个消息对象
MimeMessage message=new MimeMessage(s);
//设置邮件
InternetAddress from=new InternetAddress("testJavaWebMail@126.com");
message.setFrom(from);//设置发件人
InternetAddress to=new InternetAddress(to_mail);
message.setRecipient(Message.RecipientType.TO,to);//设置收件人,并设置其接收类型为TO
message.setSubject(to_title);//设置主题
message.setText(to_content);//设置信件内容
message.setSentDate(new Date());//设置发信时间
//发送邮件
message.saveChanges();//存储邮件信息
Transport transport=s.getTransport("smtp");
//以smtp方式登录邮箱,第一个参数是发送邮件用的邮件服务器SMTP地址,第二个参数为用户名,第
```



```
//三个参数为密码
transport.connect("smtp.126.com","testJavaWebMail","8807698");
transport.sendMessage(message,message.getAllRecipients());
//发送邮件，其中第二个参数是所有已设好的收件人地址
transport.close();
%>
<div align="center">
<p>邮件发送成功!</p>
</div>
<%
}catch(MessagingException e){
out.println("邮件发送失败!");
}
%>
</body>
</html>
```

程序的运行效果如图 11-16 所示。接着就可以到接收邮件的邮箱中去收取邮件了。

为了保证接收数据字符编码的正确性，使用了如下的语句：

```
<fmt:requestEncoding value="gb2312"/>
```

这样就将 request 中的数据字符编码设为了“gb2312”，汉字数据就可以正确地接收了。建立邮件会话的参数需要放到一个 Properties 对象中，需要设置两个参数，一是发送邮件的邮件服务器，二是进行验证的标志。

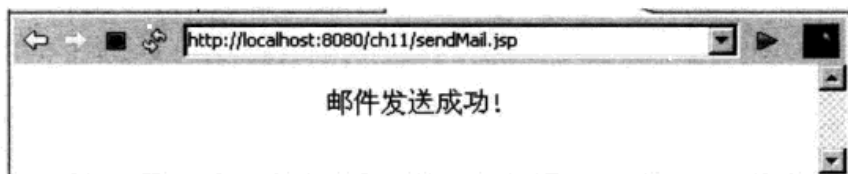


图 11-16 发送邮件

这个示例是发送文本格式的邮件，那如果是要发送 HTML 邮件呢？可以参照如下的代码段给 HTML 邮件设置邮件正文内容：

```
//-----给消息对象设置内容-----
MimeMessage message=new MimeMessage(s);
//给消息对象设置发件人、收件人、主题、发信时间（代码略，参见实例 11-1）
//新建一个存放信件内容的 BodyPart 对象
BodyPart mdp=new MimeBodyPart();
//给 BodyPart 对象设置内容和格式的编码方式
mdp.setContent(to_content,"text/html;charset=gb2312");
//新建一个 MimeMultipart 对象用来存放 BodyPart 对象(事实上可以存放多个)
Multipart mm=new MimeMultipart();
//将 BodyPart 加入到 MimeMultipart 对象中(可以加入多个 BodyPart)
mm.addBodyPart(mdp);
//把 mm 作为消息对象的内容
message.setContent(mm);
//发送邮件（代码略，参见实例 11-10）
```

代码为什么会是这样的呢？来看图 11-17。

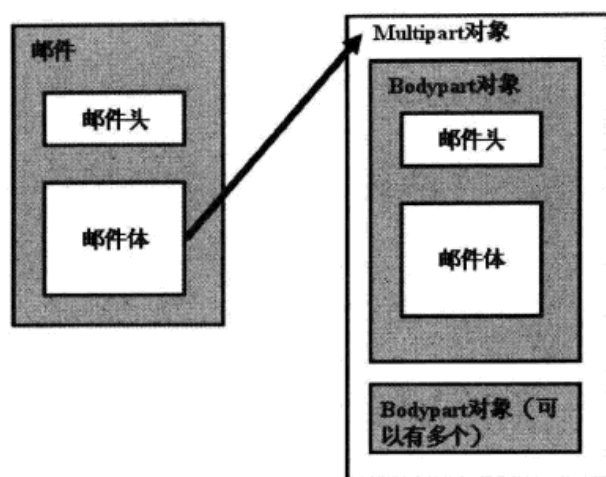


图 11-17 邮件的构成

用图对照着这段来看,先是生成了一个 `MimeBodyPart` 对象(`MimeBodyPart` 继承了 `Multipart` 对象,表示一个基于 Internet 应用的 `Multipart` 对象)并设置邮件内容,再生成了一个 `MimeMultipart` 对象,再将先生成的 `MimeBodyPart` 对象作为后一个的邮件体,再将后生成的 `MimeBodyPart` 对象作为整个邮件的邮件体,这样设置完后就可以发送了。可见,邮件的内容是可以嵌套的。

如何发送带有附件的邮件呢? 来看下面的代码。

```
MimeMessage message=new MimeMessage(s);
//给消息对象设置发件人、收件人、主题、发信时间(代码略,参见实例 11-10)
//新建一个MimeMultipart对象用来存放多个BodyPart对象
Multipart mm=new MimeMultipart();
//设置信件文本内容
BodyPart mdp=new MimeBodyPart();//新建一个存放信件内容的BodyPart对象
//给BodyPart对象设置内容和格式/编码方式
mdp.setContent(to_content,"text/html;charset=gb2312");
mm.addBodyPart(mdp);//将含有信件内容的BodyPart加入到MimeMultipart对象中
//设置信件的附件1(自定义附件:直接将所设文本内容加到自定义文件中作为附件发送)
mdp=new MimeBodyPart();//新建一个存放附件的BodyPart
DataHandler dh=new DataHandler("JavaMail 附件测试","text/plain; charset=gb2312");
//新建一个DataHandler对象,并设置其内容和格式/编码方式
mdp.setFileName("fj.txt");//加上这句将作为附件发送,否则将作为信件的文本内容
mdp.setDataHandler(dh);//给BodyPart对象设置内容为dh
mm.addBodyPart(mdp);//将含有附件的BodyPart加入到MimeMultipart对象中
//设置信件的附件2(用本地机上的文件作为附件)
mdp=new MimeBodyPart();
FileDataSource fds=new FileDataSource("d:/story.txt");
dh=new DataHandler(fds);
mdp.setFileName("story.txt");//可以和原文件名不一致
mdp.setDataHandler(dh);
mm.addBodyPart(mdp);
message.setContent(mm);//把mm作为消息对象的内容
message.saveChanges();
//发送邮件(代码略,参见实例 11-10)
```



从以上代码来看,给邮件增加附件的程序代码也是比较简单的,但是请开发人员注意,这里增加的附件都是服务器上的附件,如果是客户端的附件则需要先将文件上传到邮件服务器上再作为附件添加。向服务器上传文件的方法请参见本章的后续内容。

### 11.5.4 接收邮件

这一节的内容主要讲述如何从邮件服务上接收邮件,一般是先得到邮件列表,再显示出来,然后由用户决定查看哪封邮件的内容,再单击邮件的标题即打开此封邮件的内容。

#### 【实例 11-11】接收邮件

下面的 JSP 页面用于显示收件箱中的邮件列表。

mailList.jsp

```
<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ page import="javax.mail.*"%>
<%@ page import="javax.mail.internet.*,mail.MailUtil"%>
<html>
<head>
<title>接收邮件</title>
</head>
<body>
<table align="center" border="1">
<tr><td colspan="5" align="center">
        以下是 testJavaWebMail@126.com 邮箱中收件箱中的内容</td></tr>
<tr><td>序号</td><td>标题</td><td>发送者地址</td>
        <td>邮件大小</td><td>发送时间</td></tr>
<%
    Session mailsession=Session.getInstance(System.getProperties(),null);
    mailsession.setDebug(false);
    //protocol 为连接协议, IMAP 或是 POP
    Store store=mailsession.getStore("pop3");
    //mailhost 主机, user 为用户名, passwd 为密码
    store.connect("pop3.126.com","testJavaWebMail","8807698");
    Folder folder = store.getFolder("INBOX");
    folder.open(Folder.READ_ONLY);
    Message messages[]=folder.getMessages();
    out.println("testJavaWebMail@126.com 邮箱的收件箱中共有"+messages.length+"封邮件: <br>");
    //输出每封邮件的相关信息
    for(int i=0;i<messages.length;i++){
        out.print("<tr><td>"+(i+1)+"</td>");
        out.print("<td><a href='viewMail.jsp?id="+((MimeMessage)
            (messages[i])).getMessageID()+"'>"
            +messages[i].getSubject()+"</a></td>");
        String from =MailUtil.decodeText(messages[i].getFrom()[0].toString());
        InternetAddress ia = new InternetAddress(from);
        out.print("<td>"+ia.getAddress()+"</td>");
        out.print("<td>"+messages[i].getSize()+"</td>");
        out.print("<td>"+messages[i].getSentDate()+"</td></tr>");
    }
%>
```



```

}
folder.close(true);
store.close();
%>
</table>
</body>
</html>

```

程序运行的结果如图 11-18 所示。



图 11-18 显示邮箱收件箱中的邮件列表

单击标题将进入 viewMail.jsp 页面，显示单击的邮件的具体内容。

#### viewMail.jsp

```

<%@ page language="java" contentType="text/html; charset=gb2312"%>
<%@ page import="javax.mail.*"%>
<%@ page import="javax.mail.internet.*,mail.MailUtil"%>
<html>
<head>
<title>接收邮件</title>
</head>
<body>
<%
String messageId=request.getParameter("id");
Session mailsession=Session.getInstance(System.getProperties(),null);
mailsession.setDebug(false);
//protocol 为连接协议，IMAP 或是 POP
Store store=mailsession.getStore("pop3");
//mailhost 主机，user 为用户名，passwd 为密码
store.connect("pop3.126.com","testJavaWebMail","8807698");
Folder folder = store.getFolder("INBOX");
folder.open(Folder.READ_ONLY);
Message messages[]=folder.getMessages();
int i=0;
for(;i<messages.length;i++)
    if(((MimeMessage)(messages[i])).getMessageID().equals(messageId))
        break;
%>
<table align="center" border="1">
<tr><td>邮件标题: </td><td><%=messages[i].getSubject()%></td></tr>
<tr><td>发送者邮箱: </td><td>
<%=String from =MailUtil.decodeText(messages[i].getFrom()[0].toString());
InternetAddress ia = new InternetAddress(from);

```



```

        out.print(ia.getAddress());%>
    </td></tr>
    <tr><td>邮件内容: </td><td>
    <%
    Object content =messages[i].getContent();
    if(content instanceof String)
        out.print(content);
    if(content instanceof Multipart)
        MailUtil.dumpMultipart((Multipart)content,out);
    %>
    </td></tr>
</table>
</body>
</html>

```

查看邮件具体内容的页面运行结果如图 11-19 所示。



图 11-19 查看邮件的具体内容

程序中用到了一个 JavaBean, 这个 JavaBean 的代码如下。

#### MailUtil.java

```

package mail;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import javax.mail.MessagingException;
import javax.mail.BodyPart;
import javax.mail.Multipart;
import javax.mail.internet.MimeUtility;
import javax.servlet.jsp.JspWriter;
public class MailUtil {
    //对编码作转换
    public static String decodeText(String text) throws UnsupportedEncodingException{

```

```

        if(text==null) return null;
        if (text.startsWith("=?GB")||text.startsWith("=?gb"))
            text = MimeUtility.decodeText(text);
        else
            text = new String(text.getBytes("ISO8859_1"));
        return text;
    }
    //输出邮件内容
    public static void dumpMultipart(Multipart mmp,JspWriter out)
        throws MessagingException,IOException{
        for(int pc=0;pc<mmp.getCount();pc++){
            BodyPart bp = mmp.getBodyPart(pc);
            Object content = bp.getContent();
            if(content instanceof String){
                out.print(content);
            }else
                if(content instanceof Multipart)
                    dumpMultipart((Multipart)content,out);
                else
                    if(content instanceof InputStream)
                        out.print("邮件有附件:"+decodeText(bp.getFileName()));
            }
        }
    }
}

```

这个 **JavaBean** 中有两个方法，一个方法用于作编码转换，以正确显示发送邮件的邮箱地址；第二个方法用于递归输出邮件的内容，方法转入了两个参数，第一个是要输出的邮件内容，第二个是 **JSP** 页面的 **out** 对象。

以下的语句建立起邮件会话：

```

Session mailsession=Session.getInstance(System.getProperties(),null);
mailsession.setDebug(false);

```

以下的语句得到邮箱的 **Store** 对象：

```

Store store=mailsession.getStore("pop3");

```

接着使用服务器名称、用户名以及密码，通过调用 **Store** 的 **connect()** 方法来连接到 **Session**：

```

store.connect("pop3.126.com"," testJavaWebMail","8807698");

```

然后就可以浏览 **Folder** 的层次了，为了得到层次的顶部，要求查看一个默认的 **Folder**，可以用如下的语句：

```

folder=store.getDefaultFolder();

```

每一 **Folder** 可以包含很多其他的文件夹。例如，可以通过 **getFolder** 方法浏览收件箱。

```

Folder folder = store.getFolder("INBOX");

```

此时，可以查看 **IMAP** 树中的文件。而对于 **POP3**，它以单一队列将文件映射到一个收信箱。为了操作文件中的信息，需要打开文件夹：

```

folder.open(Folder.READ_ONLY);

```

以 **READ\_ONLY** 模式打开文件夹则表示是只读方式；而使用 **READ\_WRITE** 表示将会改变文件。下面的语句可以获得文件夹内的所有邮件：



```
Message messages[]=folder.getMessages();
```



**提示** 邮件列表默认情况下按发送者排序，如果要改成其他的排序方式，如发送日期，则需要自行编写排序算法。

邮件主题很容易获得。请注意 **Part** 接口是来自于前面的文章，即通过这一接口我们将 **BodyPart** 和 **MimeBodyPart** 构建一个信息。当解码邮件时，必须使用 **Message** 方法处理 **Parts** 的层次。处理 **MultiPart** 信息的过程相对比较复杂。**MultiPart** 意味着 **Part** 由多个部分组成，每一部分的内容必须接收到。

在显示邮件列表页面要向显示某个邮件具体内容的页面传递一个标识以唯一地标识邮件，但只有 **MimeMessage** 类可通过 **getMessageId()** 方法得到一个唯一地标识，而 **MimeMessage** 类又是 **Message** 类的子类，因此可以使用如下的方法：

```
((MimeMessage)(messages[i])).getMessageID()
```

## 11.6 小结

XML 在 Java Web 系统中经常可以见到，常用来做配置文件和交换数据。使用 DTD 和 XML Schema 可对数据规范进行约束，使用 JDOM 可快速操作 XML 文件。

使用 **jspSmartUpload** 组件来上传和下载文件是十分简便的。上传文件可以是上传到服务器的指定目录，也可以上传到数据库表中，上传到数据库中则需要将数据作为二进制数据来处理，放入到 **image** 类型的字段中。

**JavaReport** 是国产的用于开发 Web 图形与报表的工具，具有良好的实用性。生成彩色验证码可以提供系统的安全性保障，验证码不符则不进入下一步操作。

在 JDK1.6 中使用 **JavaMail** 组件只需下载和安装 **JavaMail** 组件包即可。**Session** 对象代表邮件会话，这个对象可用来创建邮件对象、实现邮件对象中数据的封装，还可指定邮件服务器认证的客户端属性；**InternetAddress** 代表邮件地址对象；**MimeMessage** 代表 MIME 风格的邮件；**Transport** 是邮件发送类；**Store** 类可用来从邮件服务器上接收邮件；**Folder** 对象代表邮件文件夹。

## 11.7 练习

1. 自行编写一个 XML 文件，内容自定，再用 JDOM 接口编写一个 JSP 程序读取这个 XML 文件的内容并显示。（程序代码可参考实例 11-2）
2. 利用自己在 Internet 邮件服务器上注册的邮箱编写一个邮件发送程序，并测试通过（代码可参考实例 11-10）。
3. 用 **JavaReport** 制作一个学生成绩统计报表，并绘制学生各成绩阶段统计饼形图（比如：90~100 为一个阶段）。
4. 编写一个用户注册程序，在注册页面生成图形验证码，并要求用户输入验证码，在接收 jsp 页面验证用户的输入是否正确。

# 12

## 基于 JSP 实现报到管理系统

在本章中将和读者一道，运用 Eclipse 开发工具来完整地体验报到管理系统的开发过程。报到管理系统比较常用，业务需求相对简单，容易理解，这样便于学习和扩展到其他系统的开发，因为不同的系统其开发过程和开发方法其实是差不多的。

读者在阅读本章时需要注意体会一个系统的整个开发过程，回味在开发过程中运用了一些什么样的软件工程思想，再自己用 Eclipse 照本章的内容逐步做一做，这样开发能力和水平就会提高得很快。

本章中使用的开发技术也很简单，系统的全部功能都会在 JSP 中实现，虽然开发起来相当简单和快速，但也会带来一些问题。先一起来开发，再总结心得和问题，在后面的章节中即可有针对性地运用各种框架技术对系统做出改进。

### 12.1 系统需求

#### 12.1.1 系统业务需求

学校在录取了新生以后，新生拿着录取通知书在开学时到学校来报到，这就需要有报到管理系统。学生报到的通用业务流程如图 12-1 所示。

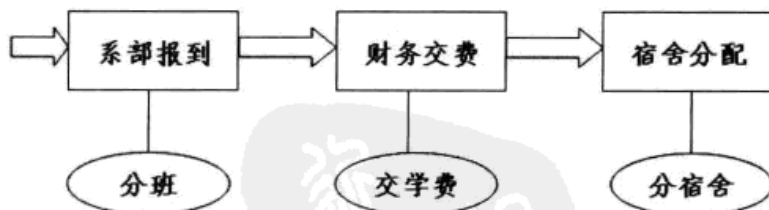


图 12-1 学生报到的通用业务流程

学生拿着学校招生办核发的录取通知书首先到系部报到，此时由系部管理人员分班，落实班级后表示此学生已报到；分班后学生到财务管理人员处交纳费用；交费后即到宿舍管理人员



处报到分配宿舍。

报到后才能交费，因为分班后即表示定了专业定了班级，不同的专业、不同的班级有可能收费标准不同；没有交费也就不能为学生分配宿舍。所以系部报到、财务交费、宿舍分配这三个过程是串联的，有先后顺序。



**提示** 事实上各个学校的学生报到流程可能有所不同，而且相对会复杂许多，比如可能会有更多的报到步骤，有些步骤是并行的，有些可能是交叉的，读者可在理解上文所述的简单基本流程的基础上再加深度学习。

### 12.1.2 系统功能需求

根据前文所述的业务需求，报到管理系统主要供学校的相关老师使用，其中系统管理人员可以使用系统所有的功能，并能进行用户管理；系部管理人员可以对学生进行分类；财务管理系人员可以对已报到的学生的收费情况进行登记；宿舍管理人员可以为已报到并已交费的学生分配宿舍。

系统的功能模块如图 12-2 所示。

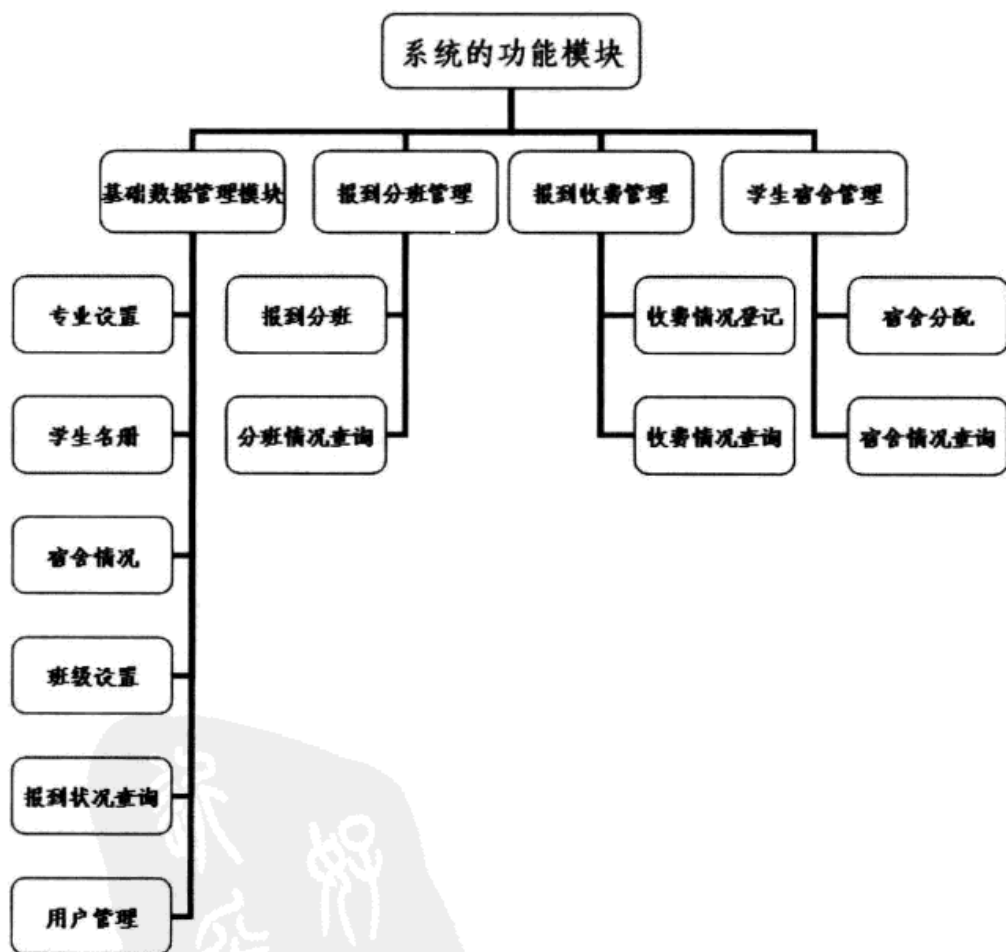


图 12-2 系统的功能模块图

系统分为基础数据管理、报到分班管理、报到收费管理、学生宿舍管理 4 个模块，此外还有一个用户登录的功能。每个用户在使用系统的各个功能之前必须先登录系统。

基础数据管理模块中的功能只有系统管理员才能使用，主要用来录入系统需要用到的一些基础数据，如开办的专业、录取的学生名册、学生宿舍的情况、班级设置的情况、所有学生报到的状况查询、系统用户的管理。报到分班管理模块供系统管理人员使用，有报到分班功能和分班情况查询功能。报到收费管理模块供财务管理人员使用，有收费情况登记功能和收费情况查询功能。学生宿舍管理模块供宿舍管理人员使用，有宿舍分配和宿舍情况查询功能。

## 12.2 系统设计

### 12.2.1 系统设计思想

系统的业务逻辑与功能需求并不复杂，这里打算采用最为简单的 JSP 编程来实现。所有的数据处理逻辑、业务逻辑、数据表现逻辑都在 JSP 页面中完成。这样做的好处是系统相对简单，开发很快；但也会带来许多问题，比如 JSP 页面中代码较多，各种逻辑代码交织在一起，维护起来相当困难，特别是在功能点较多的页面中，再比如系统的层次结构过于简单，不利于大规模团队合作开发。

系统是最为简单的二层结构，如图 12-3 所示。

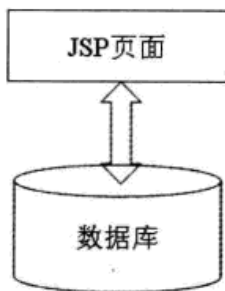


图 12-3 系统的二层设计结构

系统的所有数据放置于数据库中，这里采用的是 SQL Server。在 JSP 页面中要生成 SQL 语句，再通过 JDBC 接口将 SQL 语句发送到数据库，由数据库执行 SQL 语句，并返回执行的结果。结果返回到 JSP 页面后，再做输出处理。



**提示** 其实读者对数据库的品种不必太在意，在精通了一种数据库后，再学习其他的数据库品种的使用和管理就比较容易了。

一般地，开发人员总是先将网页界面设计出来，再与用户一起分析需求，这样就可以定下来页面跳转的流程、输入和输出元素；接下来就可以进一步开发数据库系统和 JSP 程序了。这里不展开讨论界面如何设计，这主要是美工要做的事情，但一般地，程序员也需要参与需求的讨论，并可在讨论需求的时候提前编写一些程序模块，提前进行数据库的设计。



## 12.2.2 数据库系统设计思路

在使用特定的数据库品种软件工具之前，开发人员需要先在脑海里构思数据库设计的大致情况，形成基本的图形，如果数据关系比较复杂的话，应当找来一张纸，用笔来初步勾勒出 ER 图的草图。

有的开发人员一想到数据库系统的设计就马上使用 SQL Server 来设计表，比较简单的系统可以这么做，大一些的系统如果不是经验十分丰富的数据库设计人员只怕这样很难应付。因此，笔者还是提倡大家先来构思。

如何构思呢？这时主要解决两个问题，一是大致有哪些表，二是表与表之间可能会有什么样的关系。一起来循着思路走。

要进入系统就必须有用户名和密码，而且不同的角色权限不同，因此需要有一个用户表，表中有用户名、密码、用户角色这些字段。

基础数据主要有班级、宿舍、专业、学生名册，其他还有收费的情况、分宿舍的情况、分班的情况，这些数据及数据之间的关系相对比较多。可设计班级表、宿舍表、专业表、学生表。

一般地，一个班级可以有多名学生，而一名学生只属于一个班级，因此班级表与学生表是“多对一”的关系，“多对一”的关系在“多”这边体现，也就是说，在学生表中放置外键“班级 ID 号”，关联到班级表的主键“班级 ID 号”。

宿舍表、专业表与学生表的关系，就像班级表与学生表的关系一样，是“多对一”的关系，都需要在学生表中放置外键来体现这种关系，即“宿舍 ID 号”、“专业 ID 号”。

收费情况如何体现呢？一名学生对对应着他自己的收费情况，不存在“一对多”的情况，那么“一对一”的情况要怎么办呢？直接将收费情况的字段放置于学生表中即可。分宿舍的情况和分班的情况如何体现呢？学生在没有分配宿舍和没有分班时，外键“宿舍 ID 号”、“班级 ID 号”这些都是空值，如有则表示分配了。

以上数据表及数据之间的关系如图 12-4 所示。

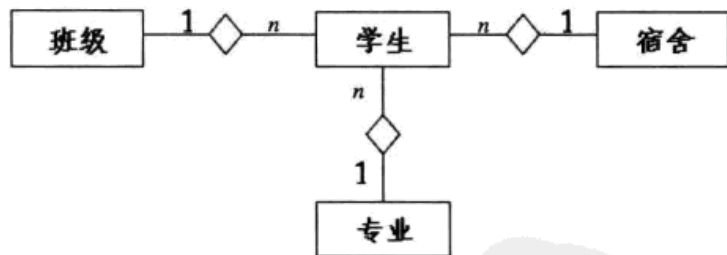


图 12-4 初步构思的 ER 草图



**提示** 现实情况中也许会更加复杂一些，比如学生入校后还可以转专业，就有转专业操作的功能，再比如班级还会与专业进行关联，因为班级是属于某个专业的。

好了，数据表及数据之间关系现在基本理清了，还有就是如何来控制系部报到、财务交费、宿舍分配这个串行的业务流程呢？小型的系统总不可能还去开发一个工作流引擎吧。是的，为

简单起见，可以让程序中的逻辑与数据一起来配合完成业务流程。

报到时做分班处理，会向学生表中的“班级 ID 号”字段里填数据，表示已分班，在未分班之前，程序应控制不允许做收费登记和分配宿舍。收费登记仅允许对已分班的学生登记收费情况，收到多少钱、是否交清这些情况填到学生表的相关字段中，只有交清了学费的学生才能分配宿舍。这就相当于对学生的报到状况做了记录，在报到过程中有不同的状态，状态之间的转换遵循一定的规则，如图 12-5 所示。

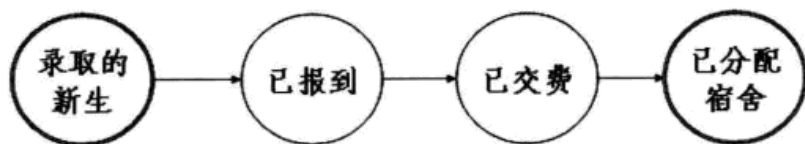


图 12-5 报到过程中的状态

这种状态之间的转换规则并不复杂，如果在相当复杂的情况下，应当用专门的字段来标识当前所处的状态，用配置文件、某种数据结构或专门的一个状态转换类来标示转换的规则。这里为简单起见，直接根据相关字段值的情况来判断学生报到的状况。

### 12.2.3 数据库系统的实现

下面一起来用 SQL Server 实现数据库系统。在 SQL Server 2008 中首先打开“SQL Server Management Studio”，在“连接到服务器”对话框中输入用户名和密码，然后单击“确定”按钮，连接到数据库系统。

#### ① 创建数据库

在“SQL Server Management Studio”左边的树形菜单中选中“数据库”节点，单击右键，在弹出的快捷菜单中选择“新建数据库(N)...”，即会弹出“新建数据库”对话框，如图 12-6 所示

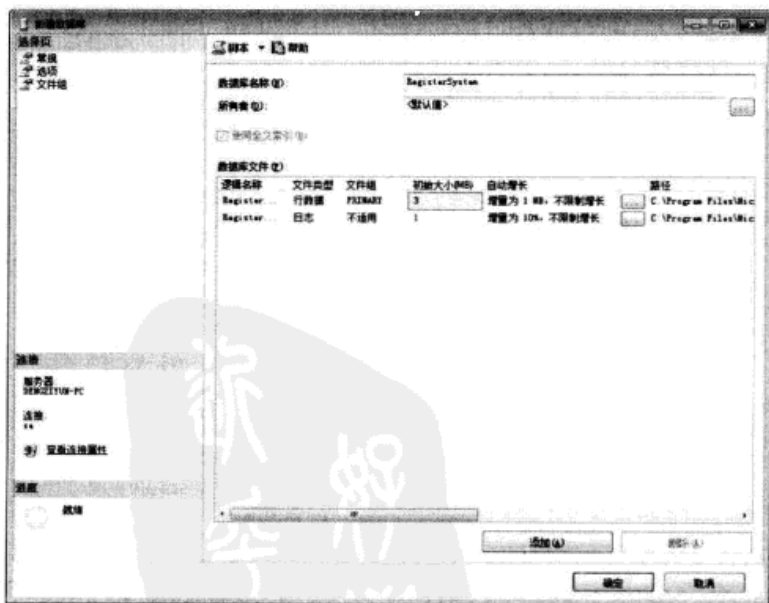



图 12-6 创建 RegisterSystem 数据库



在数据库“名称”后的输入框中输入数据库的名称，此处设为“RegisterSystem”，单击“确定”按钮，完成数据库的创建。SQL Server 用起来特别方便，许多选项会自动为用户考虑，因此在简单的应用场合下使用默认选项即可。

## ② 创建数据表

在这一步中要依次建立学生表 (Student)、用户表 (adminuser)、班级表 (ClassTa)、专业表 (Speciality) 和宿舍表 (Bedchamber)。

在“SQL Server Management Studio”左边的树形菜单中选中“RegisterSystem”数据库的“表”节点，单击右键，弹出快捷菜单，选择“新建表(N)...”，弹出新建表对话框，如图 12-7 所示。在中间的表格编辑器中设计数据结构。StudentId 字段设为 numeric 数据类型，自动增量，设为关键字。设定为关键字的操作方法是单击 StudentId 字段设计的那一行，使光标落下，再单击按钮栏的“”按钮，即可将 StudentId 字段设为关键字。

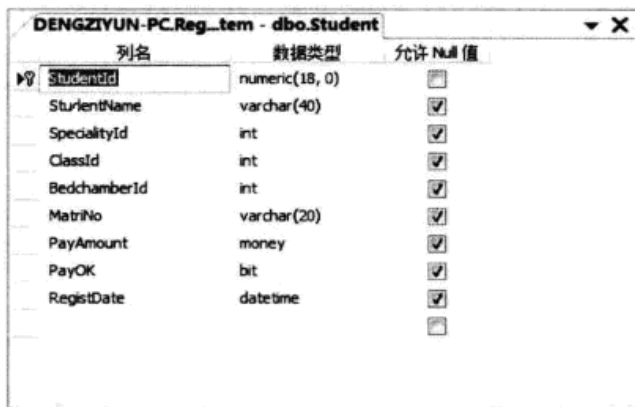



图 12-7 建立学生表

按图 12-7 所示编辑完 Student 表的数据结构后，单击“”按钮即弹出“选择名称”对话框，在其中输入表的名称 Student，单击“确定”按钮，完成 Student 表的创建。

读者也可以使用 SQL Server 查询分析的查询分析器以执行 SQL 语句的方式来创建 Student 表，语句如下：

----创建 Student 表----

```
CREATE TABLE [dbo].[Student] (
    [StudentId] [numeric](18, 0) IDENTITY (1, 1) NOT NULL ,
    [StudentName] [varchar] (40) COLLATE Chinese_PRC_CI_AS NULL ,
    [SpecialityId] [int] NULL ,
    [ClassId] [int] NULL ,
    [BedchamberId] [int] NULL ,
    [MatriNo] [varchar] (20) COLLATE Chinese_PRC_CI_AS NULL ,
    [PayAmount] [money] NULL ,
    [PayOK] [bit] NULL ,
    [RegisDate] [datetime] NULL
) ON [PRIMARY]
GO
```

----将 Student 表的主键设为 StudentId 字段----

```
ALTER TABLE [dbo].[Student] ADD
    CONSTRAINT [PK_Student] PRIMARY KEY CLUSTERED
    (
```

```

        [StudentId]
    ) ON [PRIMARY]
GO

```



**提示** 以上语句是笔者使用 SQL Server 语句生成器自动生成的语句,对于初学者建议使用企业管理器来操作。[dbo]表示表的所有者是 dbo,这是系统管理员的代称,在默认情况下,sa 就是 dbo。

建立用户表的步骤与学生表的步骤相同,不再赘述,给出表的各列设计情况,如图 12-8 所示。

列名	数据类型	允许 Null 值
adminusername	varchar(40)	<input type="checkbox"/>
adminuserpassword	varchar(40)	<input checked="" type="checkbox"/>
adminuserrole	int	<input checked="" type="checkbox"/>

图 12-8 建立用户表

读者也可使用 SQL 语句来创建用户表,语句如下:

```

----创建 adminuser 表----
CREATE TABLE [dbo].[adminuser] (
    [adminusername] [varchar] (40) COLLATE Chinese_PRC_CI_AS NOT NULL ,
    [adminuserpassword] [varchar] (40) COLLATE Chinese_PRC_CI_AS NULL ,
    [adminuserrole] [int] NULL
) ON [PRIMARY]
GO
----设置 adminuser 表的主键为 adminusername 字段----
ALTER TABLE [dbo].[adminuser] ADD
    CONSTRAINT [PK_adminuser] PRIMARY KEY CLUSTERED
    (
        [adminusername]
    ) ON [PRIMARY]
GO

```

建立班级表的步骤与学生表的步骤相同,不再赘述,给出表的各列设计情况,如图 12-9 所示。

列名	数据类型	允许 Null 值
ClassId	int	<input type="checkbox"/>
ClassName	varchar(40)	<input checked="" type="checkbox"/>

图 12-9 建立班级表



相应的建表 SQL 语句如下：

```

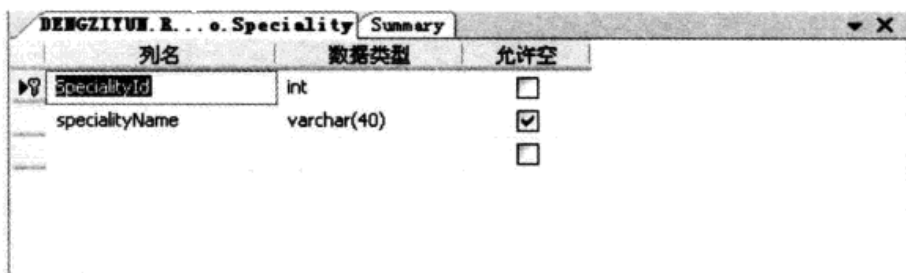
----创建 ClassTa 表----
CREATE TABLE [dbo].[ClassTa] (
    [ClassId] [int] IDENTITY (1, 1) NOT NULL ,
    [ClassName] [varchar] (40) COLLATE Chinese_PRC_CI_AS NULL
) ON [PRIMARY]
GO
----设置 ClassTa 表的主键为 ClassId 字段----
ALTER TABLE [dbo].[ClassTa] ADD
    CONSTRAINT [PK_ClassTa] PRIMARY KEY CLUSTERED
    (
        [ClassId]
    ) ON [PRIMARY]
GO

```



**提示** 班级表的名称为什么不使用 Class 呢？因为 Java 语句中将 class 设为关键字，为避免混淆故将班级表的表名设为 ClassTa。

建立专业表的步骤与学生表的步骤相同，不再赘述，给出表的各列设计情况，如图 12-10 所示。



列名	数据类型	允许空
SpecialityId	int	<input type="checkbox"/>
specialityName	varchar(40)	<input checked="" type="checkbox"/>

图 12-10 建立专业表

相应的建表 SQL 语句如下：

```

----创建 Speciality 表----
CREATE TABLE [dbo].[Speciality] (
    [SpecialityId] [int] IDENTITY (1, 1) NOT NULL ,
    [specialityName] [varchar] (40) COLLATE Chinese_PRC_CI_AS NULL
) ON [PRIMARY]
GO
----设置 Speciality 表的主键为 SpecialityId 字段----
ALTER TABLE [dbo].[Speciality] ADD
    CONSTRAINT [PK_Speciality] PRIMARY KEY CLUSTERED
    (
        [SpecialityId]
    ) ON [PRIMARY]
GO

```

建立宿舍表的步骤与学生表的步骤相同，不再赘述，给出表的各列设计情况，如图 12-11 所示。



图 12-11 建立宿舍表

相应的建表 SQL 语句如下：

```

----创建 Bedchamber 表----
CREATE TABLE [dbo].[Bedchamber] (
    [BedchamberId] [int] IDENTITY (1, 1) NOT NULL ,
    [BedchamberName] [varchar] (40) COLLATE Chinese_PRC_CI_AS NULL
) ON [PRIMARY]
GO
----设置 Bedchamber 表的主键为 BedchamberId 字段----
ALTER TABLE [dbo].[Bedchamber] ADD
    CONSTRAINT [PK_Bedchamber] PRIMARY KEY CLUSTERED
    (
        [BedchamberId]
    ) ON [PRIMARY]
GO

```

### ③ 设置表与表之间的关系

数据表设计好了，下面来设置表与表之间的关联关系。从数据库系统设计的思路来看，所有数据表以学生表为核心，构建数据关联关系，为简化操作，可从学生表进入操作，如图 12-12 所示。



图 12-12 从学生表进入设置关系操作



在企业管理器的树形菜单中选中“表”节点下的 Student 表，单击右键，弹出快捷菜单，选择“设计”选项，进入 Student 的设计对话框。在表的设计对话框中单击任意处，使光标落在编辑区，单击右键，弹出快捷菜单，选择“关系(H)...”选项，进入“外键关系”对话框，如图 12-12 所示。

单击“添加(A)”按钮，会自动生成一个关系，选中该关系可编辑属性。单击“表和列规范”前的“+”号展开，在“表和列规范”右侧会出现一个“...”按钮，并出现“表和列”对话框，如图 12-13 所示。

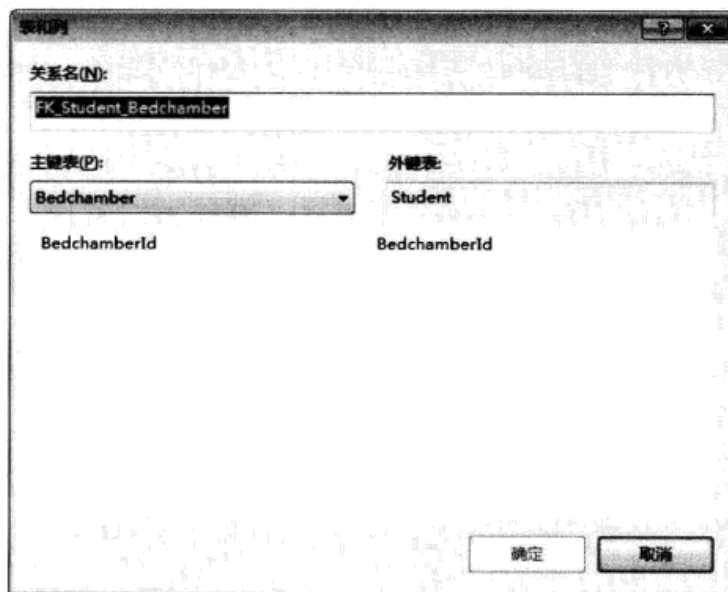



图 12-13 “表和列”对话框

设置完成后单击“关闭”按钮，再单击按钮栏中的“”按钮，保存关系的设置情况。设置表与表之间的关系也可以使用如下的 SQL 语句。

```
ALTER TABLE [dbo].[Student] ADD
CONSTRAINT [FK_Student_Bedchamber] FOREIGN KEY
(
    [BedchamberId]
) REFERENCES [dbo].[Bedchamber] (
    [BedchamberId]
),
CONSTRAINT [FK_Student_ClassTa] FOREIGN KEY
(
    [ClassId]
) REFERENCES [dbo].[ClassTa] (
    [ClassId]
),
CONSTRAINT [FK_Student_Speciality] FOREIGN KEY
(
    [SpecialityId]
) REFERENCES [dbo].[Speciality] (
    [SpecialityId]
)
GO
```

## 12.3 系统开发框架搭建

### 12.3.1 在 Eclipse 中搭建 Web 应用的开发框架

使用 Eclipse 的优点就是可以方便地搭建起 Web 应用的开发框架，而不必手工烦琐地创建文件夹、文件。

打开 Eclipse 后，选择“File”→“New”→“Project...”菜单，会弹出“New Project”对话框。在“New Project”对话框中间的树形菜单中选择“Dynamic Web Project”节点，单击“Next”按钮，进入到“New Dynamic Web Project”对话框，如图 12-14 所示。

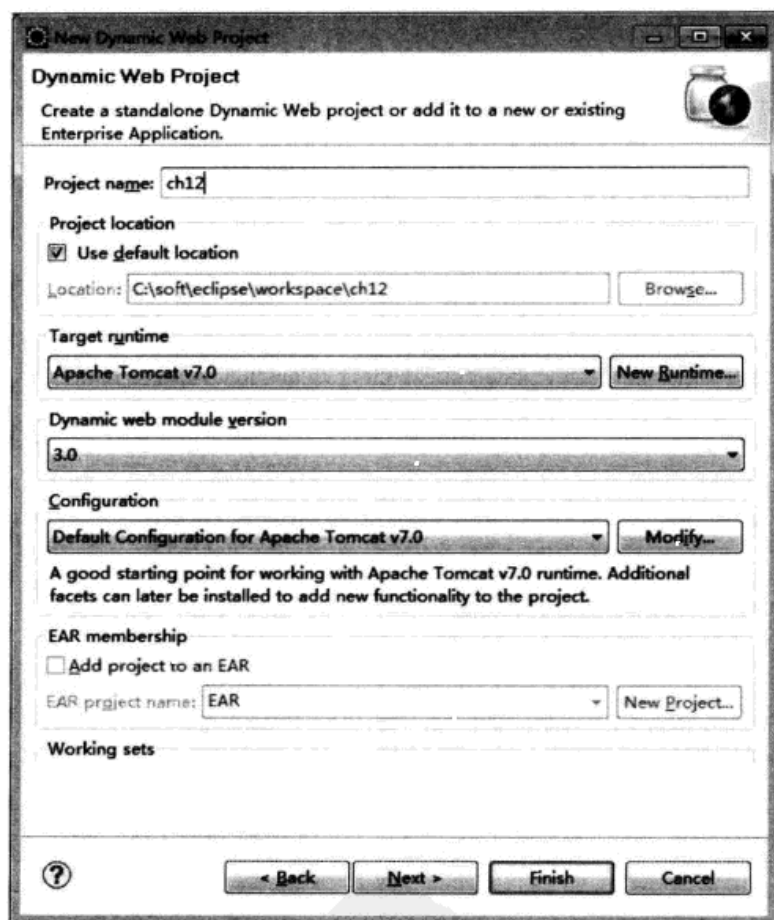


图 12-14 搭建 Web 应用的开发框架

在“New Dynamic Web Project”对话框中输入工程名“ch12”，在“Project location”下设置工程文件放置的路径。在默认情况下会使用默认的设置，即为 Eclipse 工作区（workspace）的工作目录。不选中“Use default location”前的复选框，即可使用下面的路径文本框和“Browse”按钮来设置工程文件放置的目录，此时可以不采用系统的默认设置。

在“Target runtime”下设置 Web 应用的目标运行时环境，此处为 Apache Tomcat v7。单击“Finish”按钮完成 Web 应用框架的搭建。搭建完成的工程情况如图 12-15 所示。





图 12-15 Web 应用框架搭建完成

从图 12-15 中可以看出，已经将 JDK 和 Tomcat 自带的组件包都作为当前工程的组件包。build 目录中将存放编辑后的类的字节码文件。WebContent 是自动生成的存放 Web 应用文件的目录，Eclipse 已自动搭建好了 Web 应用的结构，包括“WEB-INF”文件夹，web.xml 文件，这就是使用 Eclipse 的好处，能够自动帮助程序员做许多事，而不必手工创建了。

### 12.3.2 设计总体的页面效果

为方便用户使用 Web 系统，常采用框架技术来搭建总体的页面效果，如图 12-16 所示。

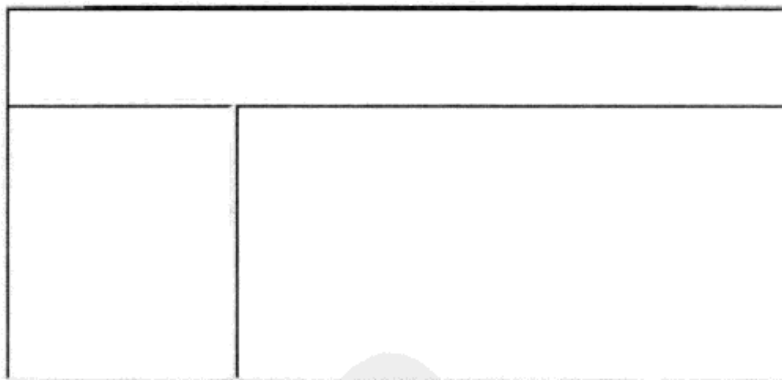


图 12-16 总体的页面效果

实际上相当于将一个网页拆分为三个部分，每部分又是一个网页。上面的部分用来放置系统的名称、广告条等内容。左边的部分常用于放置用户操作菜单，然后单击菜单中的某项菜单即会在右边展现出操作的内容。

图 12-16 是一个静态的网页，实际工程中可以改成 JSP 页面作为首页，还可以将边框的宽度设为 0，这样用户就感觉不到是使用框架技术将网页分成了三个部分。系统最终开发的效果如图 12-17 所示。

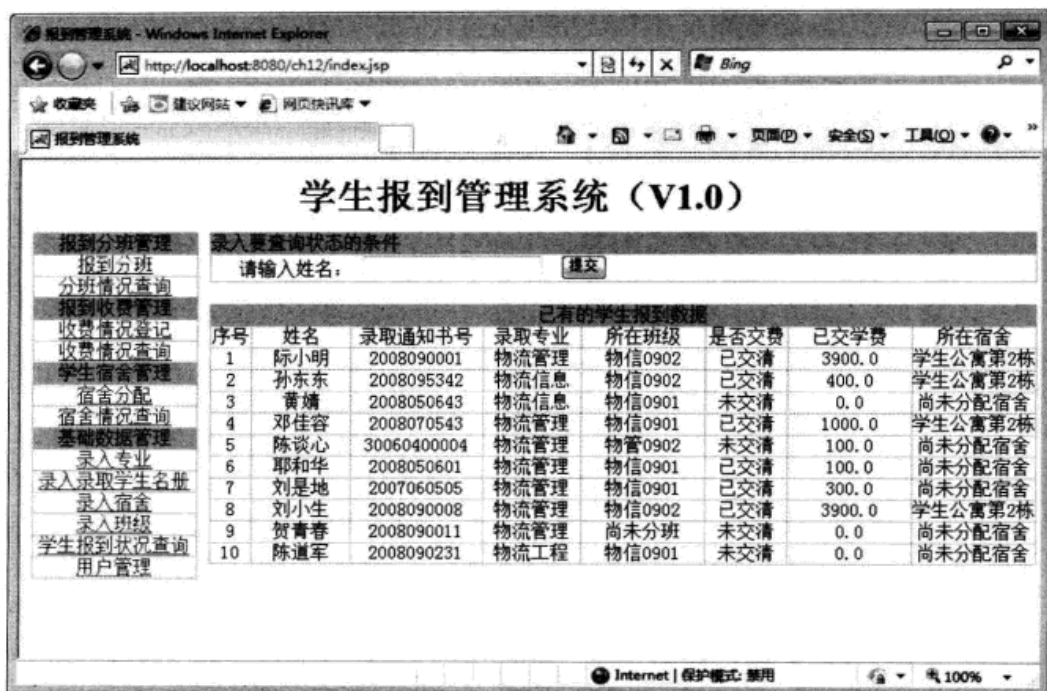


图 12-17 系统的设计效果

一起来看看 index.jsp 的源代码就明白了。

#### index.jsp

```
<%@ page contentType="text/html; charset=GBK" %>
<%if(session.getAttribute("adminusername")==null||session.getAttribute(
    "adminusername").toString().length()==0)
    response.sendRedirect("login.jsp");
%>
<html>
<head>
<title>报到管理系统</title>
</head>
<frameset framespacing="0" border="0" frameborder="0" rows="64,*"
    marginwidth="0" marginheight="0">
    <frame name="banner" scrolling="no" noresize target="contents"
        src="banner.html">
    <frameset cols="160,*">
        <frame name="left" target="main" scrolling="auto" src="left.jsp"
            marginwidth="10" marginheight="0">
        <frame name="main" scrolling="auto" marginwidth="0" marginheight="0"
            src="basicdata/regstatus.jsp">
    </frameset>
</frameset>
<noframes>
<body>
</body>
</noframes>
</frameset>
</html>
```

index.jsp 页面中使用了如下的语句做会话检查。



```
<%if(session.getAttribute("adminusername")==null||session.getAttribute(
    "adminusername").toString().length()==0)
    response.sendRedirect("login.jsp");
%>
```

首先是检查 session 变量 adminusername 的内容是否为空,如果是则表示用户还没有登录,将页面重定向到 login.jsp 页面要求用户登录。而在 login.jsp 页面中只要用户登录了就会在 session 中设置 session 变量 adminusername 的值为登录的用户名。

<frameset>标签中嵌套了三个<frame>标签。名为 banner 的<frame>标签即为页面顶部的内容,通过 src 属性来设置内容为 banner.html 页面。banner.html 页面的源代码如下:

banner.html

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    <div align="center">
        <font size="6"><strong>学生报到管理系统 (V1.0) </strong></font>
    </div>
</body>
</html>
```

第二个<frame>标签设置为左边的树形菜单,通过 src 属性来设置内容为 left.jsp 页面。left.jsp 页面的源代码如下:

left.jsp

```
<%@ page contentType="text/html; charset=GBK" %>
<html>
<head><title>登录系统</title></head>
<body>
<div align="center">
<table border="1" cellpadding="0" cellspacing="0"
    style="border-collapse: collapse" bordercolor="#C0C0C0" width="140">
<%String adminuserrole=session.getAttribute("adminuserrole")+"";
    int adminuserroleint=0;
    if(adminuserrole!=null&&adminuserrole.length()!=0)
        adminuserroleint=Integer.parseInt(adminuserrole);
%>
<%if(adminuserroleint==1||adminuserroleint==2){ %>
    <tr>
        <td width="100%" bgcolor="#C0C0C0" align="center">
            <font color="#0000FF">报到分班管理</font>
        </td>
    </tr>
    <tr>
        <td width="100%" align="center">
            <font><a href="class/classadmin.jsp" target="main">
                报到分班</a></font>
        </td>
```



```

</tr>
<tr>
  <td width="100%" align="center">
    <font><a href="class/classview.jsp" target="main">
      分班情况查询</a></font>
    </td>
  </tr>
</tr>
<%> %>
<%if(adminuserroleint==1||adminuserroleint==3){ %>
  <tr>
    <td width="100%" bgcolor="#C0C0C0" align="center">
      <font color="#0000FF">报到收费管理</font>
    </td>
  </tr>
  <tr>
    <td width="100%" align="center">
      <font><a href="money/acceptmoney.jsp" target="main">
        收费情况登记</a></font>
      </td>
    </tr>
    <tr>
      <td width="100%" align="center">
        <font><a href="class/classview.jsp" target="main">
          收费情况查询</a></font>
        </td>
      </tr>
    <%> %>
    <%if(adminuserroleint==1||adminuserroleint==4){ %>
      <tr>
        <td width="100%" bgcolor="#C0C0C0" align="center">
          <font color="#0000FF">学生宿舍管理</font>
        </td>
      </tr>
      <tr>
        <td width="100%" align="center">
          <font><a href="bed/bedchamber.jsp" target="main">
            宿舍分配</a></font>
          </td>
        </tr>
        <tr>
          <td width="100%" align="center">
            <font><a href="class/classview.jsp" target="main">
              宿舍情况查询</a></font>
            </td>
          </tr>
        <%> %>
        <%if(adminuserroleint==1){ %>
          <tr>
            <td width="100%" bgcolor="#C0C0C0" align="center">
              <font color="#0000FF">基础数据管理</font>
            </td>
          </tr>
        </tr>

```



```

<tr>
  <td width="100%" align="center">
    <font><a href="basicdata/specialityadmin.jsp" target="main">
      录入专业</a></font>
    </td>
  </tr>
</tr>
<tr>
  <td width="100%" align="center">
    <font><a href="basicdata/matri.jsp" target="main">
      录入录取学生名册</a></font>
    </td>
  </tr>
</tr>
<tr>
  <td width="100%" align="center">
    <font><a href="basicdata/bedchamber.jsp" target="main">
      录入宿舍</a></font>
    </td>
  </tr>
</tr>
<tr>
  <td width="100%" align="center">
    <font><a href="basicdata/class.jsp" target="main">录入班级</a></font>
    </td>
  </tr>
</tr>
<tr>
  <td width="100%" align="center">
    <font><a href="basicdata/regstatus.jsp" target="main">
      学生报到状况查询</a></font>
    </td>
  </tr>
</tr>
<tr>
  <td width="100%" align="center">
    <font><a href="basicdata/adminuser.jsp" target="main">
      用户管理</a></font>
    </td>
  </tr>
</tr>
<%} %>
</table>
</div>
</body>
</html>

```

这就是系统的功能菜单。其间通过读取 session 变量 adminuserrole 的值来获取用户的角色（如果用户登录成功即会在 session 中记录 adminuserrole 变量值为用户角色，详见 12.4.1 节“用户登录功能的实现”的内容），根据用户角色来相应地显示可操作的功能菜单。

那么又是如何让单击菜单后就可更新右边的内容呢？可以一起来看看操作菜单的超链接设置，举例如下：

```
<a href="basicdata/regstatus.jsp" target="main">学生报到状况查询</a>
```

<a>标签中的 target 属性指出了目标框架的名称，也就是说单击“学生报到状况查询”文字链接后会更新 main 框架中的内容，内容为 href 属性中设置的页面。

## 12.4 系统各功能点的实现

下面一起来实现每个功能点。读者在阅读时可先参看前面几个功能是如何实现的，跟着一起来操作，在操作了前面的 2~3 个功能点后，后面的功能点的实现直接阅读就能看明白了，再进行操作就会比较熟练和快速了。

### 12.4.1 用户登录功能的实现

用户登录功能的实现要做以下的操作：开发一个登录页面 `login.jsp`，用户访问此页面时可输入用户名和密码，再单击“提交”按钮，即在页面中做数据检查，如果用户名和密码正确则导向 `index.jsp` 页面，如果不正确则在 `login.jsp` 页面报错。

因为 `login.jsp` 页面是要建在 Web 应用的根目录中，也即工程的 `WebContent` 目录中，故先选中 Eclipse 左边树形菜单中的“`WebContent`”，单击右键，弹出快捷菜单，选择“`New`”→“`JSP`”菜单，弹出“`New JavaServer Page`”对话框，在“`File name`”文本框中输入文件名“`login.jsp`”，然后单击“`Finish`”按钮，完成 `login.jsp` 的创建工作，如图 12-18 所示。

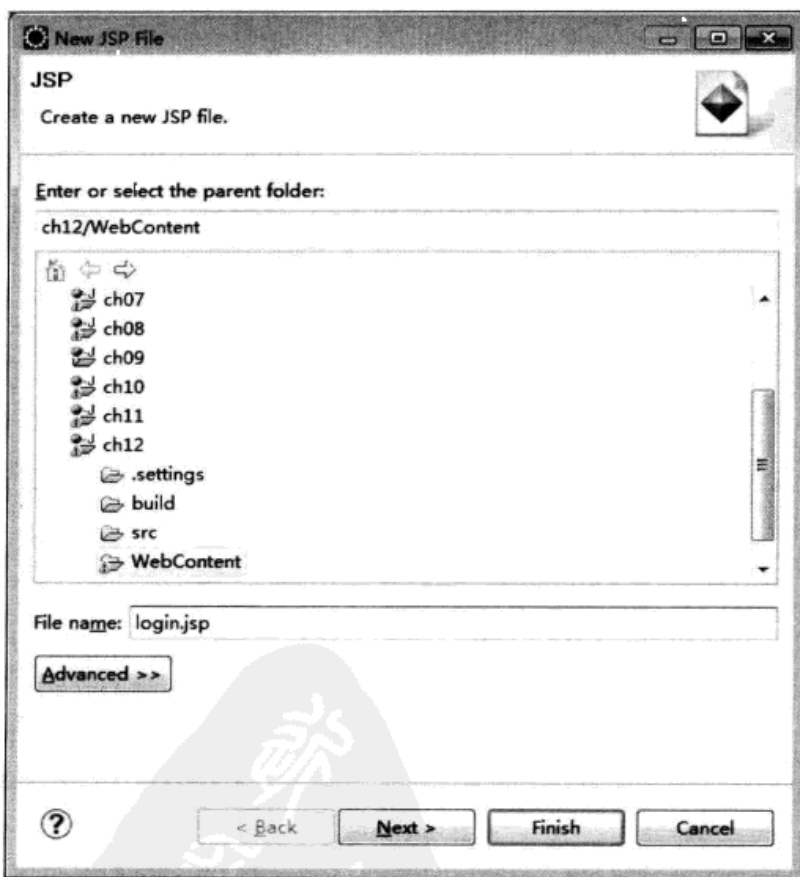


图 12-18 新建 `login.jsp` 页面

在 Eclipse 的代码编辑区中修改 `login.jsp` 的源代码，修改后的代码如下所示。





```

</tr>
</table>
</form>
</div>
</body>
</html>
<%DBConn.closeConn(conn);%>

```

从源代码中的以下行：

```
<form method="post" action="login.jsp">
```

可以得知，表单以 **post** 方式被提交到 **login.jsp** 页面，即本页面。表单项中有一个名为 **action** 的隐藏域，值为 **login**，标明了一旦表单提交则表示是做登录操作。在 **login.jsp** 页面的一段 JSP 代码中将接收 **action** 参数的值，再作出判断是否做登录操作。如果是则通过构造 SQL 语句查询，如果结果记录集中有记录则表示用户名和密码正确无误，再在 **session** 中记下用户名和用户的角色，并将页面重定向到 **index.jsp** 页面。

为防止 SQL 注入攻击，使用了 **PreparedStatement** 对象来执行带参数的 SQL 语句。如果校验没有通过，则将错误的提示文件放入到字符串 **errmsg** 中，在后续的代码显示表格时，如果 **errmsg** 的值不为空，说明有错误消息，则显示出来。

页面的执行结果如图 12-19 所示。

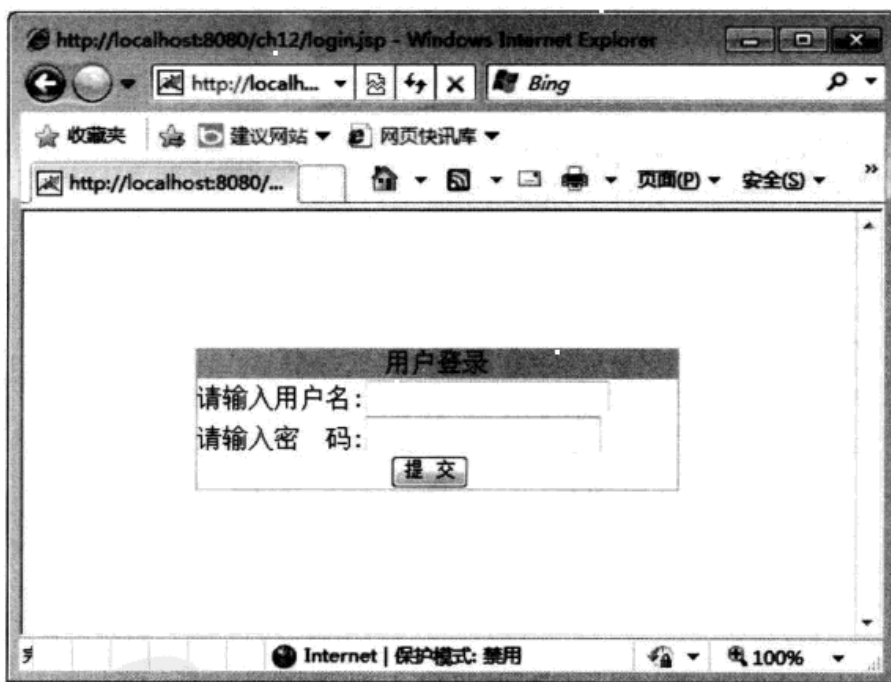


图 12-19 用户登录操作

**login.jsp** 页面中用到了 **csai.db.DBConn** 类，这个类用于生成一个数据库连接对象。**DBConn** 类属于 **csai.db** 包，因此开发时需要先创建 **csai.db** 包，然后在这个包中创建 **DBConn** 类。

在 Eclipse 左边的树形菜单中选“**Java Resources:src**”，单击右键，弹出快捷菜单，选择“**New**”→“**Package**”，弹出“**New Java Package**”对话框。在“**Name**”文本框中输入“**csai.db**”，然后单击“**Finish**”按钮，完成“**csai.db**”包的创建，如图 12-20 所示。



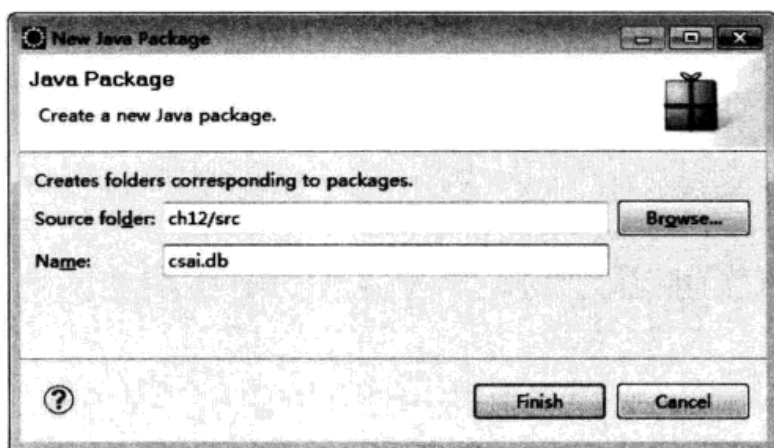


图 12-20 创建 csai.db 包

“csai.db”包创建完成后，即可以在“Java Resources:src”下看到“csai.db”节点了。接下来再创建 DBConn 类。

在 Eclipse 左边的树形菜单中选中“csai.db”节点，单击右键，弹出快捷菜单，选择“New”→“Class”菜单，弹出“New Java Class”对话框。在“Name”文本框中输入类名“DBConn”，然后单击“Finish”按钮，完成 DBConn 类的创建，如图 12-21 所示。最后在 Eclipse 的代码编辑器中修改 DBConn 类的源代码，修改后代码如下。



图 12-21 创建 DBConn 类

DBConn.java

```

package csai.db;
import java.sql.Connection;
import java.sql.DriverManager;
public class DBConn {
    //得到数据库连接
    public static Connection createDBConn(){
        try{
            Connection conn=DriverManager.getConnection(
                "jdbc:sqlserver://127.0.0.1:1433;
                DatabaseName=RegisterSystem","sa","123");
            return conn;
        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }
    //关闭数据库连接
    public static void closeConn(Connection conn){
        try{
            conn.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}

```

这个类中有两个静态的方法，一个是 `createDBConn()`，用于生成一个数据库连接对象；另一个是 `closeConn()`，用于关闭一个数据库连接。因此在 `login.jsp` 页面的顶部和底部分别使用了如下的语句：

```

<%Connection conn=DBConn.createDBConn(); %>
<%DBConn.closeConn(conn);%>

```

前面的语句生成了一个数据库连接对象 `conn`，在后续代码中会使用到这个对象；后面的语句关闭前面生成的数据库连接对象 `conn`。

## 12.4.2 专业基础数据管理功能的实现

专业基础数据管理功能要实现对专业表 (Speciality) 的增加、删除操作，因字段相当简单，没有必要提供修改操作。一般地，专业设置这种基础数据在系统开始运行时就会录入这些数据，并轻易不作改动，因此功能应当由系统管理员来使用。功能操作的界面如图 12-22 所示。

单击左边的“录入专业”命令就会进入到专业基础数据管理功能界面。进入后会以表格的形式显示已有的专业数据，在每条显示记录的后面有一个“删除”超链接，单击即可删除此专业。在“录入专业数据”表格中，输入专业的名称，再单击“提交”按钮，即可增加一个专业。

为区分模块间的程序，可以将基础数据管理模块中的 JSP 页面放置于一个专门的文件夹 `basicdata` 中。

因为是要在 Web 应用下创建 `basicdata` 文件夹，因此在 Eclipse 左边的树形菜单中选中



WebContent 节点，单击右键，弹出快捷菜单，选择“New”→“Folder”菜单，弹出“New Folder”对话框。在“Folder name”文本框中输入“basicdata”，然后单击“Finish”按钮，完成“basicdata”文件夹的创建，如图 12-23 所示。

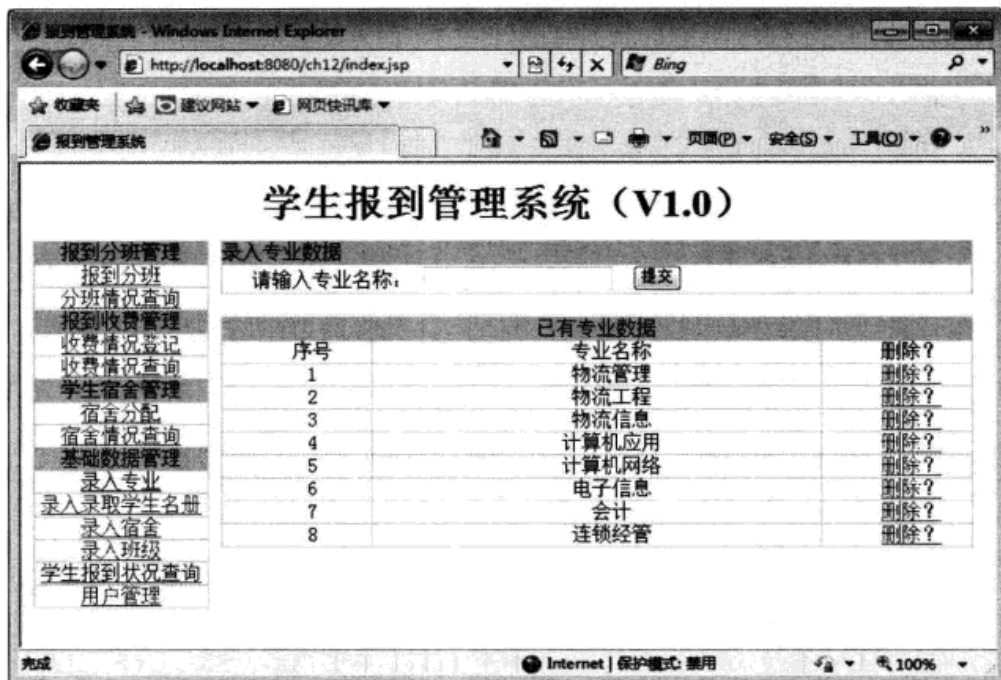


图 12-22 专业基础数据管理功能界面

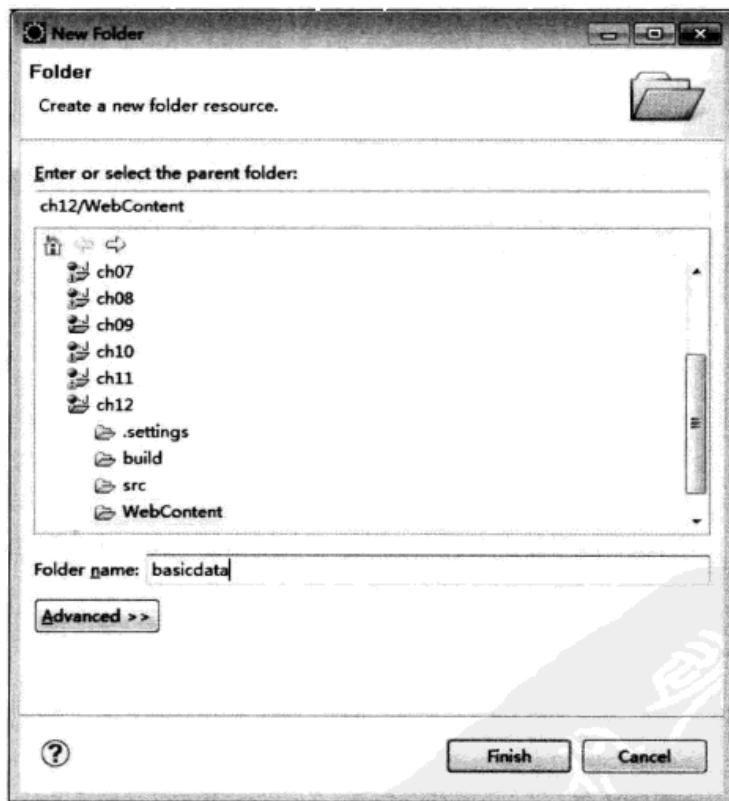


图 12-23 新建 basicdata 文件夹

接着在 basicdata 文件夹中创建 specialityadmin.jsp 文件，修改其中的内容，源代码如下。

#### specialityadmin.jsp

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page import="csai.db.DBConn,java.sql.*" %>
<%Connection conn=DBConn.createDBConn(); %>
<html>
<body>
<%
    //----获取请求参数值----
    String action=request.getParameter("action");
    if(action!=null&&action.length()!=0)
        action=new String(action.getBytes("ISO-8859-1"));
    String specialityname=request.getParameter("specialityname");
    if(specialityname!=null&&specialityname.length()!=0)
        specialityname=new String(specialityname.getBytes("ISO-8859-1"));
    String specialityid=request.getParameter("specialityid");
    if(specialityid!=null&&specialityid.length()!=0)
        specialityid=new String(specialityid.getBytes("ISO-8859-1"));
    //----如果是要增加一个专业---
    if("add".equals(action)){
        String sql="select * from Speciality where SpecialityName=?";
        PreparedStatement preSQLSelect=conn.prepareStatement(sql);
        preSQLSelect.setString(1,specialityname);
        ResultSet rs=preSQLSelect.executeQuery();
        if(!rs.next()){//没有这个专业
            sql="insert into Speciality(SpecialityName) values(?)";
            PreparedStatement preSQLInsert=conn.prepareStatement(sql);
            preSQLInsert.setString(1,specialityname);
            preSQLInsert.executeUpdate();
        }
    }
    //----如果是删除一个专业----
    if("del".equals(action)){
        String sql="delete from Speciality where SpecialityId=?";
        PreparedStatement preSQLDel=conn.prepareStatement(sql);
        int specialityidInt=0;
        if(specialityid!=null&&specialityid.length()>0)
            specialityidInt=Integer.parseInt(specialityid);
        preSQLDel.setInt(1,specialityidInt);
        preSQLDel.executeUpdate();
    }
%>
<form method="post" action="specialityadmin.jsp">
<table border="1" cellpadding="0" cellspacing="0"
    style="border-collapse:collapse" bordercolor="#C0C0C0" width="600">
<tr>
<td width="100%" bgcolor="#C0C0C0">
<font color="#0000FF">录入专业数据</font></td>
</tr>
<tr>
<td width="100%">
    请输入专业名称:

```



```




</td>
</tr>
</table>
</form>


|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |  |  |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
| <td width="20%" align="center">       序号     </td>     <td width="60%" align="center">       专业名称     </td>     <td width="20%" align="center">       删除?     </td>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |  |  |
| <%         //----查询出已有的专业数据----         String sql="select * from Speciality";         Statement state=conn.createStatement();         ResultSet rs=state.executeQuery(sql);         int i=0;         while(rs.next()){           i++;         %>       <tr>         <td width="20%" align="center">           <%=i%>         </td>         <td width="60%" align="center">           <%=rs.getString("SpecialityName")%>         </td>         <td width="20%" align="center">           <a href="specialityadmin.jsp?action=del&specialityid=<%=rs.getInt("SpecialityId")%>">             删除? </a>           </td>         </tr>       <% } %>     </table>   </body> </html>   <%DBConn.closeConn(conn);%> |  |  |


```

在表单中使用了一个名为 `action` 的隐藏域，值为 `add`，表明是要增加一个专业；在删除的

超链接中也带了 `action` 参数, 值为 `del`, 表示是要删除一个专业。在页面的第一个 JSP 程序块中, 首先是接收传来的参数。为了解决中文乱码的问题, 对提交的数据作了编码转换, 比如在增加一个专业的操作中, 传来的参数 `specialityname` 中可能会有中文, 因此使用了如下的语句作编码转换:

```
if(specialityname!=null&&specialityname.length()!=0)
    specialityname=new String(specialityname.getBytes("ISO-8859-1"));
```

此后再使用数据就会是正常的了。根据 **action** 值的判断，如果值为 **add**，则做增加一个专业的操作。先是构造 **SQL** 语句做 **select** 查询，看专业表中是否已存在同名的专业，如有则不必再增加，如无再构造插入操作的 **SQL** 语句，并做更新处理。

根据 `action` 值的判断，如果值为 `del`，则做删除一个专业的操作。先是删除操作构造 SQL 语句，再做更新处理。

### 12.4.3 录取学生名册基础数据管理功能的实现

录取学生名册基础数据管理功能的操作界面如图 12-24 所示。为集成地在一个 JSP 页面中进行录入、查询、删除操作，操作界面也设计得相对复杂一些。



图 12-24 录取学生名册基础数据管理功能的操作界面

上面的“录入录取学生名册”表格用于录入学生名册中的学生数据，要录入的数据主要有该学生的姓名、录取通知书号及录取的专业。学生姓名和录取通知书号以文本框的形式录入，录取专业以下拉框的形式录入。录入完毕后，单击“提交”按钮，将数据提交到当前页面，即做在学生表中增加记录的操作处理。

在下面的“查询已录入的学生名册”表格中，上半部分用于输入查询的条件。查询条件根据学生姓名和录取的专业进行组合，单击“提交”按钮，即可将数据提交到当前页面，再相应



地构造 SQL 语句, 查询出结果。

这里对查询结果还做了分页处理, 当转向其他页时, 需要带入已有的查询条件, 因此要将查询参数带入到超链接参数中。为构造出仅查询出当前页数据的复杂 SQL 语句, 程序员会巧妙地采取一些办法来构造 SQL 语句。一起来看这个页面 `matri.jsp` 的程序源代码, 再来做分析。

`matri.jsp`

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page import="csai.db.DBConn,java.sql.*" %>
<%Connection conn=DBConn.createDBConn(); %>
<html>
<body>
<% //----获取请求参数值----
String action=request.getParameter("action");
if(action!=null&&action.length()!=0)
    action=new String(action.trim().getBytes("ISO-8859-1"));
String specialityid=request.getParameter("specialityid");
if(specialityid!=null&&specialityid.length()!=0)
    specialityid=new String(specialityid.trim().getBytes("ISO-8859-1"));
String matrino=request.getParameter("matrino");
if(matrino!=null&&matrino.length()!=0)
    matrino=new String(matrino.trim().getBytes("ISO-8859-1"));
String studentname=request.getParameter("studentname");
if(studentname!=null&&studentname.length()!=0)
    studentname=new String(studentname.trim().getBytes("ISO-8859-1"));
String currentpage=request.getParameter("currentpage");
if(currentpage!=null&&currentpage.length()!=0)
    currentpage=new String(currentpage.trim().getBytes("ISO-8859-1"));
//----如果是要增加一个学生---
if("add".equals(action)){
    String sql="select * from Student where matrino=? and studentname=?"+
        " and specialityid=?";
    PreparedStatement preSQLSelect=conn.prepareStatement(sql);
    preSQLSelect.setString(1,matrino);
    preSQLSelect.setString(2,studentname);
    preSQLSelect.setString(3,specialityid);
    ResultSet rs=preSQLSelect.executeQuery();
    if(!rs.next()){//没有这个专业
        sql="insert into Student(matrino,studentname,specialityid) "+
            " values(?,?,?)";
        PreparedStatement preSQLInsert=conn.prepareStatement(sql);
        preSQLInsert.setString(1,matrino);
        preSQLInsert.setString(2,studentname);
        preSQLInsert.setInt(3,Integer.parseInt(specialityid));
        preSQLInsert.executeUpdate();
    }
}
//----如果是要删除一个学生---
if("del".equals(action)){
    String sql="delete from Student where studentid=?";
    PreparedStatement preSQLUpdate=conn.prepareStatement(sql);
    preSQLUpdate.setString(1,request.getParameter("studentid"));
    preSQLUpdate.executeUpdate();
}
```



```

}
//----如果是要查询数据----
ResultSet rsselect=null;
int pagesize=20;//每页记录条数
int pagecount=0;//总页数
int currentpageint=1;//当前页码
int recount=0;//总记录条数
if(currentpage!=null&&currentpage.length()!=0)
    currentpageint=Integer.parseInt(currentpage);
if("select".equals(action)){
    String sql=null;
    if((specialityid==null||specialityid.length()==0)&&(
        studentname==null||studentname.length()==0)){
        sql="select top "+pagesize+" Student.studentname as studentname,"+
            "Student.studentid as studentid,"+
            "Student.matrino as matrino,"+
            "Speciality.specialityname as specialityname "+
            "from Student,Speciality where "+
            "Student.specialityid=Speciality.specialityid ";
        String sqlcount="select count(*) as recount "+
            "from Student,Speciality where "+
            "Student.specialityid=Speciality.specialityid ";
        Statement stateCount=conn.createStatement();
        ResultSet rscount=stateCount.executeQuery(sqlcount);
        rscount.next();
        recount=rscount.getInt("recount");//得到总记录条数
        //----得到总页数----
        if(recount%pagesize==0)//能整除
            pagecount=recount/pagesize;
        else//不能整除
            pagecount=(int)(recount/pagesize)+1;
        //----生成得到当前页数据查询的附件条件----
        if(pagecount>1&&currentpageint>1){
            String sqladd=" and studentid not in "+
                "(select top "+(currentpageint-1)*pagesize+
                " Student.studentid as studentid "+
                "from Student,Speciality where "+
                "Student.specialityid=Speciality.specialityid "+
                "order by studentid desc) ";
            sql=sql+sqladd;
        }
        sql=sql+" order by studentid desc ";
        Statement state=conn.createStatement();
        rsselect=state.executeQuery(sql);
    }else{
        int specialityidInt=0;
        if(specialityid!=null&&specialityid.length()>0)
            specialityidInt=Integer.parseInt(specialityid);
        //----先生成查询的 where 子句----
        String sqlwhere=" where Student.specialityid=
            Speciality.specialityid";
        if(specialityidInt!=0)//如果选择了专业
            sqlwhere=sqlwhere+" and Student.specialityid=? "+

```



```

        "and Student.studentname like ? ";
    else//如果没有选择专业
        sqlwhere=sqlwhere+" and Student.studentname like ? ";
    sql="select top "+pagesize+" Student.studentname as studentname,"+
        "Student.studentid as studentid,"+
        "Student.matrino as matrino,"+
        "Speciality.specialityname as specialityname "+
        "from Student,Speciality "+sqlwhere;
    //----得到总记录条数----
    String sqlcount="select count(*) as recount "+
        "from Student,Speciality "+sqlwhere;
    PreparedStatement preSQLCount=conn.prepareStatement(sqlcount);
    if(specialityidInt!=0){//如果选择了专业
        preSQLCount.setInt(1,specialityidInt);
        preSQLCount.setString(2,"%"+studentname+"%");
    }else
        preSQLCount.setString(1,"%"+studentname+"%");
    ResultSet rscount=preSQLCount.executeQuery();
    rscount.next();
    recount=rscount.getInt("recount");//得到总记录条数
    //----得到总页数----
    if(recount%pagesize==0)//能整除
        pagecount=recount/pagesize;
    else//不能整除
        pagecount=(int)(recount/pagesize)+1;
    //----生成得到当前页数据查询的附件条件----
    PreparedStatement preSQLSelect=null;
    //总页数大于1且当前页码大于1
    if(pagecount>1&&currentpageint>1){
        String sqladd=" and studentid not in "+
            "(select top "+(currentpageint-1)*pagesize+
            " Student.studentid as studentid "+
            "from Student,Speciality "+sqlwhere+
            "order by studentid desc) ";
        sql=sql+sqladd;
        sql=sql+" order by studentid desc ";
        preSQLSelect=conn.prepareStatement(sql);
        if(specialityidInt!=0){//如果选择了专业
            preSQLSelect.setInt(1,specialityidInt);
            preSQLSelect.setString(2,"%"+studentname+"%");
            preSQLSelect.setInt(3,specialityidInt);
            preSQLSelect.setString(4,"%"+studentname+"%");
        }else{//如果没有选择专业
            preSQLSelect.setString(1,"%"+studentname+"%");
            preSQLSelect.setString(2,"%"+studentname+"%");
        }
    }
    }else{//当前为第1页或仅只有1页
        sql=sql+" order by studentid desc ";
        preSQLSelect=conn.prepareStatement(sql);
        if(specialityidInt!=0){//如果选择了专业
            preSQLSelect.setInt(1,specialityidInt);
            preSQLSelect.setString(2,"%"+studentname+"%");
        }else
            preSQLSelect.setString(1,"%"+studentname+"%");
    }
}

```



```

        }
        rsselect=preSQLSelect.executeQuery();
    }
}

%>
<form method="post" action="matri.jsp">
<table border="1" cellpadding="0"
    cellspacing="0" style="border-collapse:collapse"
    bordercolor="#C0C0C0" width="600">
    <tr>
        <td width="100%" bgcolor="#C0C0C0">
            <font color="#0000FF">录入录取学生名册</font></td>
        </tr>
        <tr>
            <td width="100%" align="left">
                请输入学生姓名:
                <input type="text" name="studentname">
                请选取录取专业:
            <%//----查询出专业数据----
                String sql="select * from Speciality";
                Statement state=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
                ResultSet rs=state.executeQuery(sql);
            %>
                <select name="specialityid">
                    <option value="">==请选择==</option>
                <%while(rs.next()){ %>
                    <option value="<%=rs.getInt("specialityid")%>">
                        <%=rs.getString("specialityname")%>
                    </option>
                <%} %>
                </select><br>
                请输入录取通知书号: <input type="text" name="matrino">
                <input type="hidden" name="action" value="add">
                <input type="submit" value="提交">
            </td>
        </tr>
    </table>
</form>
<form method="post" action="matri.jsp">
<table border="1" cellpadding="0" cellspacing="0"
    style="border-collapse:collapse"
    bordercolor="#C0C0C0" width="600">
    <tr>
        <td width="100%" bgcolor="#C0C0C0" colspan="5">
            <font color="#0000FF">查询已录入的学生名册</font></td>
        </tr>
        <tr>
            <td colspan="5">
                请输入学生姓名:
                <input type="text" name="studentname">
                请选取录取专业:
            <%//----查询出专业数据----
                sql="select * from Speciality";
            %>
                <select name="specialityid">
                    <option value="">==请选择==</option>
                <%while(rs.next()){ %>
                    <option value="<%=rs.getInt("specialityid")%>">
                        <%=rs.getString("specialityname")%>
                    </option>
                <%} %>
                </select><br>
                请输入录取通知书号: <input type="text" name="matrino">
                <input type="hidden" name="action" value="add">
                <input type="submit" value="提交">
            </td>
        </tr>
    </table>
</form>

```



```

rs.beforeFirst();
%>
<select name="specialityid">
  <option value="">==请选择==</option>
<%while(rs.next()){ %>
  <option value="<%=rs.getInt("specialityid")%>">
    <%=rs.getString("specialityname")%>
  </option>
<%} %>
</select>
<input type="hidden" name="action" value="select">
<input type="submit" value="提交">
</td>
</tr>
<tr>
  <td align="center" colspan="5"><font color="#0000FF">
    <%if(pagecount>1&&currentpageint>1){%>
      <a href="matri.jsp?currentpage=1&action=<%=action%>
        &specialityid=<%=specialityid%>
        &studentname=<%=studentname%>">首页</a>&nbsp;
      <a href="matri.jsp?currentpage=<%=currentpageint-1%>&action=<%=action%>
        &specialityid=<%=specialityid%>
        &studentname=<%=studentname%>">上一页</a>&nbsp;
    <%} %>
    <%if(pagecount>1&&currentpageint<pagecount){%>
      <a href="matri.jsp?currentpage=<%=currentpageint+1%>&action=<%=action%>
        &specialityid=<%=specialityid%>
        &studentname=<%=studentname%>">下一页</a>&nbsp;
      <a href="matri.jsp?currentpage=<%=pagecount%>&action=<%=action%>
        &specialityid=<%=specialityid%>
        &studentname=<%=studentname%>">尾页</a>&nbsp;
    <%} %>
    共<%=recount%>条记录, 共<%=pagecount %>页, 当前第<%=currentpageint%>页
  </font></td>
</tr>
<tr bgcolor="#C0C0C0">
  <td align="center"><font color="#0000FF">序号</font></td>
  <td align="center"><font color="#0000FF">姓名</font></td>
  <td align="center"><font color="#0000FF">录取专业</font></td>
  <td align="center"><font color="#0000FF">录取通知书号</font></td>
  <td align="center"><font color="#0000FF">删除? </font></td>
</tr>
<%
int i=0;
while(rsselect!=null&&rsselect.next()){
  i++;%>
  <tr>
    <td align="center"><%=i%></td>
    <td align="center">
      <%=rsselect.getString("studentname") %>
    </td>
    <td align="center">
      <%=rsselect.getString("specialityname") %>

```



```

        </td>
        <td align="center">
            <%=rsselect.getString("matrino") %>
        </td>
        <td align="center">
            <a href="matri.jsp?action=del&studentid=<%=rsselect.getString
                ("studentid")%>">删除</a>
        </td>
    </tr>
<%} %>
</table>
</form>
</body>
</html>
<%DBConn.closeConn(conn);%>

```



**提示** 像 `matri.jsp` 这样的 JSP 页面代码相当冗长，编辑经验不是特别丰富的开发人员乍一看代码都会看晕头，可想而知做系统维护的维护人员要如何来修改代码了。因此，在后续章节中还会采用一些框架技术来简化 JSP 页面的代码，而尽量地将一些业务逻辑、数据逻辑封装起来。

一起来逐步分析这些代码，就会全面理清这些程序，这有助于读者编写更复杂的系统，以及大大增强程序开发的能力。

在获取请求参数值的代码块中，获取了这些参数的值：`action`、`specialityid`、`matrino`、`studentname`、`currentpage`。`action` 参数表示要进行的操作，值为 `add` 表示要增加一名学生的数据到录取名册中；值为 `del` 表示要从名册中删除一名学生的数据；值为 `select` 表示是查询数据。

当 `action` 参数的值为 `add` 时，表单中应当也会提交过来 `specialityid`、`matrino`、`studentname` 这些参数，分别表示专业 ID 号、录取通知书号、学生姓名。再根据这 3 个参数来构造 SQL 语句，查询 `Student` 表中是否已经存在满足以下 SQL 语句查询的记录：

```
select * from Student where matrino=? and studentname=? and specialityid=?
```

再使用 `PreparedStatement` 语句对象，分别将接收到的 `matrino`、`studentname`、`specialityid` 参数值填充到对应的位置，再做查询。如果结果记录集中没有记录则表示目前名册中还没有这名学生，需要做增加处理，于是再构造进行 `insert` 操作的 SQL 语句，如下所示：

```
insert into Student(matrino,studentname,specialityid) values(?,?,?)
```

再次使用 `PreparedStatement` 语句对象填入参数值，执行更新操作。

当 `action` 参数的值为 `del` 时，表单中会提交 `studentid` 参数，这个参数表示要删除的学生 ID 号，根据这个 ID 号构造如下的 SQL 语句：

```
delete from Student where studentid=?
```

使用 `PreparedStatement` 语句对象填入参数值后即可使用语句对象的 `executeUpdate()` 方法做更新处理。

当 `action` 参数的值为 `select` 时，由于要做分页处理，程序逻辑相对会复杂一些。如果是通过提交表单的形式来做查询则不会传来当前页的页码，因此默认设置为 1；如果是通过超链接



提交参数的形式来做查询则会传来当前页的页码。此外还会传递过来如下的参数：  
studentname、specialityid。这两个参数分别表示学生姓名、录取专业的 ID 号，根据这两个参  
数的值来构造 SQL 语句。那么又如何做到仅查询出当前页的数据呢？基本原理如图 12-25 所示。



图 12-25 查询出当前页数据的原理示意图

从图 12-25 中可以看出：

当前页 = (当前页 + 当前页之前的页) - 当前页之前的页

读者可能要说了，这叫什么公式？也太简单了吧。是的，别急，待笔者细细解说。在 SQL Server 中提供了“select top n”语句，也就是说查询出每个表中满足条件的前 n 条记录，那么如果当前页为第 2 页，每页记录条数为 10 条，则当前页的数据为：

“select top 2\*10” - “select top (2-1)\*10”

就是这个原理了，“select top 2\*10”取出了第 1 页和第 2 页的数据，“select top (2-1)\*10”取出了第 1 页的数据，则两者之差为当前页数据。然而，要作集合的减法运算也是一个复杂的运算，因为首先就需要将记录集查询出来，实际上有些操作是重复的，比如如果当前页之前的数据相当多，则要耗费服务器较多的内存和做运算处理。

为降低运算的复杂性，提高运算速度，可以使用 select 语句的“not in”子句来处理。假设某表的 id 字段为关键字，表名为 tablename，查询的 where 子句条件为 sqlwhere，当前页为第 n 页，每页记录条数为 pagesize，则可以使用如下的表述来说明这个思路：

```
select top pagesize * from tablename
where sqlwhere and id not in
    (select top (n-1)*pagesize id from tablename
     where sqlwhere order by id asc
    )
order by id asc
```

这样构造出来的 SQL 语句查询速度就相当的快，因为集合运算时仅针对主键做处理，主键的查找效率是相当之高的，而且由于在子查询中只取出主键，数据量也较少，占用的内存空间也会比较低。

这里要注意以下两点：

(1) 如果当前页页码值小于或等于 1 则不必生成“not in”子句，因为子句中的(n-1)\*pagesize 值为 0 或负数是不必要的或不可取的。

(2) sqlwhere 查询条件、排序的字段、排序的方式、要查询的表的表名要在主 select 语句和“not in”子句中的 select 语句中保持一致，这样可保证查询结果的准确性。

下面一起来看程序中是如何做的。首先使用了如下的语句设置了一些初始参数：

```
int pagesize=20;//每页记录条数
int pagecount=0;//总页数
int currentpageint=1;//当前页码
int recount=0;//总记录条数
if(currentpage!=null&&currentpage.length()!=0)
    currentpageint=Integer.parseInt(currentpage);
```

要想得到数据的总页数就需要先用一个“select count(\*)”语句来得到总记录条数。因此构造了如下的 SQL 语句：

```
String sqlcount="select count(*) as recount "+
    "from Student,Speciality where "+
    "Student.specialityid=Speciality.specialityid ";
```

如果有传递过来姓名和专业 ID 号，则 where 子句会更复杂一些，

```
//----先生成查询的 where 子句----
String sqlwhere=" where Student.specialityid=Speciality.specialityid";
if(specialityidInt!=0)//如果选择了专业
    sqlwhere=sqlwhere+" and Student.specialityid=? "+
    "and Student.studentname like ? ";
else//如果没有选择专业
    sqlwhere=sqlwhere+" and Student.studentname like ? ";
//----得到总记录条数的 SQL 语句----
String sqlcount="select count(*) as recount "+
    "from Student,Speciality "+sqlwhere;
```

然后再做 SQL 查询得到满足条件的记录总条数。为保证用学生姓名可做模糊查询，采用 like 子句，但是在 PreparedStatement 对象中，需要采用如下的方法来设置模糊查询参数的值：

```
preSQLCount.setString(1,"%"+studentname+"%");
```

在得到总记录条数后，即可得到总页数，计算方法如下：

```
//----得到总页数----
if(recount%pagesize==0)//能整除
    pagecount=recount/pagesize;
else//不能整除
    pagecount=(int)(recount/pagesize)+1;
```

构造查询当前页的 SQL 语句的方法就不再赘述了，只要读者能理解以上的基本原理与方法，相信理解后续的语句并不难。

#### 12.4.4 其他基础数据管理功能的实现

宿舍基础数据管理、班级基础数据管理功能的实现方法 and 专业基础数据管理功能的实现方法相同，也就不再重复叙述了，读者可在读懂“12.4.2 专业基础数据管理功能的实现”一节的基础上再直接查看宿舍基础数据管理、班级基础数据管理功能的实现源代码，就会很容易清楚了。



### 12.4.5 学生报到状况查询功能的实现

学生报到查询的操作界面如图 12-26 所示。

单击左边菜单中的“学生报到状况查询”超链接即可进入学生报到查询界面。在查询条件中输入学生的姓名，单击“提交”按钮，即会在下面的表格中查询出所要查询的学生的报到状况。报到的状况包括分班情况、交费情况、所在宿舍的情况。此处功能比较简单，为节省篇幅不再列出源代码，读者可参看随书光盘中的源代码。

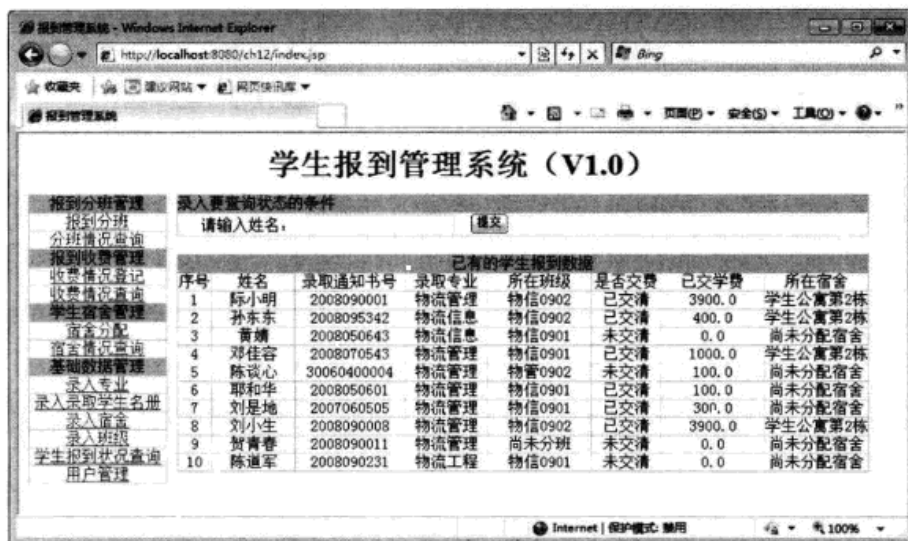


图 12-26 学生报到查询操作界面

### 12.4.6 用户管理功能的实现

用户管理功能的操作界面如图 12-27 所示。

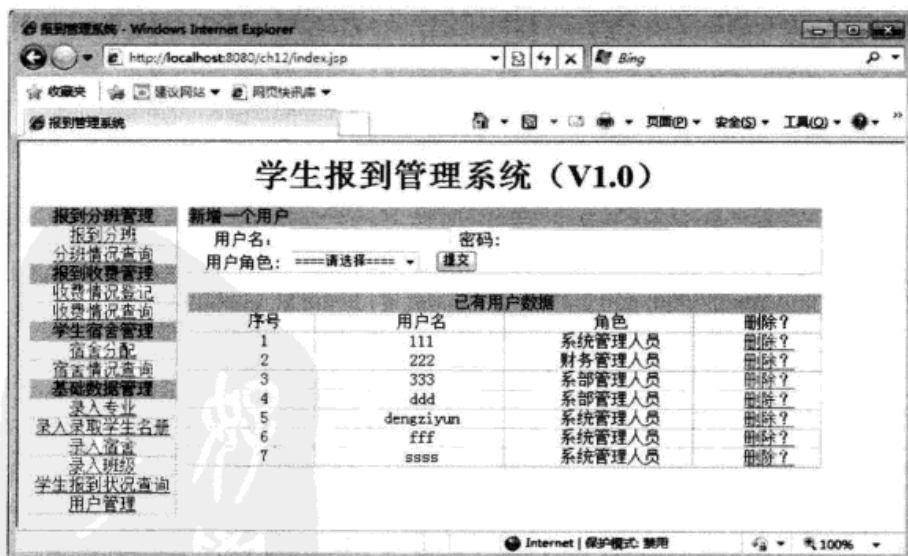


图 12-27 用户管理功能操作界面

单击左边菜单中的“用户管理”超链接，即可进入用户管理功能操作界面。在上面的“新增一个用户”表格中输入用户名、密码、用户角色，单击“提交”按钮即可新增一个用户。在下面的已有用户数据表格中列出了目前系统已有的所有用户数据，每条记录后有一个“删除？”超链接，单击即可删除当前行用户。此处功能比较简单，为节省篇幅不再列出源代码，读者可参看随书光盘中的源代码。

## 12.4.7 报到分班功能的实现

报到分班功能的操作界面如图 12-28 所示。单击左边菜单中的“报到分班”超链接即可进入到报到分班功能操作界面。上面的表格用于输入查询条件。当学生前来报到时，主要提供两个信息：姓名和录取通知书号。输入其中的任意一个或都输入即可查询出满足条件的记录，如果都不输入则表示查询出所有的记录。



图 12-28 报到分班功能操作界面

查询出所要分班的学生数据后，有可能有多条记录。在每条记录后给出一个班级的下拉列表，用于设置当前行表示的学生所在的班级。设置完查询出的学生的分班情况后，单击“确定”按钮即可。

这里有一个难点，查询结果的记录条数是不确定的，那么表单中输入项的个数相应地也就不能确定，那么又要如何对接收的分班数据并进行分班处理呢？看看这个页面的源代码就清楚了。

```
classadmin.jsp
<%@ page contentType="text/html; charset=GBK" %>
<%@ page import="csai.db.DBConn, java.sql.*" %>
<%Connection conn=DBConn.createDBConn(); %>
<%
    //-----获取请求参数值-----
```



```

String studentname=request.getParameter("studentname")+"";
if(studentname!=null&&studentname.length()!=0)
    studentname=new String(studentname.getBytes("ISO-8859-1"));
String action=request.getParameter("action")+"";
String matrino=request.getParameter("matrino")+"";
//----构造查询的SQL语句----
String sqlwhere=new String("");
String sql=new String("");
if("select".equals(action)){//如果是查询操作
    if(studentname!=null&&studentname.trim().length()!=0)
        sqlwhere="where studentname like '%" +studentname.trim()+"%' ";
    if(sqlwhere!=null&&sqlwhere.length()!=0){
        if(matrino!=null&&matrino.trim().length()!=0)
            sqlwhere+=" and matrino like '%" +matrino.trim()+"%' ";
    }else{
        if(matrino!=null&&matrino.trim().length()!=0)
            sqlwhere=" where matrino like '%" +matrino.trim()+"%' ";
    }
    sql="select * from student "+sqlwhere;
}
//----设置分班情况----
if("update".equals(action)){//如果是设置分班操作
    String studentcount=request.getParameter("studentcount");
    for(int i=1;i<=Integer.parseInt(studentcount);i++){
        String studentid=request.getParameter("studentid"+i);
        String classid=request.getParameter("classid"+i);
        if(classid!=null&&classid.length()!=0&&studentid!=
            null&&studentid.length()!=0){
            String sqlstr="update student set classid="+classid+
                "where studentid="+studentid;
            Statement state=conn.createStatement();
            state.executeUpdate(sqlstr);
        }
    }
    out.print("设置分班操作成功!");
}

%>
<html>
<body>
<form method="post" action="classadmin.jsp">
<table border="1" cellpadding="0" cellspacing="0"
    style="border-collapse:collapse" bordercolor="#C0C0C0" width="700">
<tr>
<td width="100%" bgcolor="#C0C0C0">
<font color="#0000FF">要查询的条件</font></td>
</tr>
<tr>
<td width="100%">
请输入姓名:
<input type="text" name="studentname">
请输入录取通知书号:
<input type="text" name="matrino">
<input type="hidden" name="action" value="select">
<input type="submit" value="提交">

```



```

        </td>
    </tr>
</table>
</form>
<form action="classadmin.jsp" method="post">
<table border="1" cellpadding="0" cellspacing="0"
    style="border-collapse:collapse"
    bordercolor="#C0C0C0" width="700">
    <tr>
        <td width="100%" bgcolor="#C0C0C0" align="center" colspan="8">
            <font color="#0000FF">查询到的学生数据</font></td>
        </tr>
        <tr>
            <td width="5%" align="center">序号</td>
            <td width="12%" align="center">姓名</td>
            <td width="16%" align="center">录取通知书号</td>
            <td width="12%" align="center">录取专业</td>
            <td width="15%" align="center">所在班级</td>
            <td colspan="3"></td>
            <td colspan="2"></td>
        </tr>
        <%if("select".equals(action)){%>
        <%    Statement state=conn.createStatement();
            ResultSet rs=state.executeQuery(sql);
            int i=0;
            while(rs.next()){
                i++;
            %>
            <tr>
                <td width="5%" align="center"><%=i%></td>
                <td width="12%" align="center"><%=rs.getString("studentname") %></td>
                <td width="16%" align="center"><%=rs.getString("matrino") %></td>
                <td width="12%" align="center">
                <%int specialityid=rs.getInt("specialityid");
                    if(specialityid==0)
                        out.print("尚无专业");
                    else{
                        sql="select * from speciality where specialityid="+specialityid;
                        Statement statetemp=conn.createStatement();
                        ResultSet rstemp=statetemp.executeQuery(sql);
                        if(rstemp.next())
                            out.print(rstemp.getString("specialityname"));
                        else
                            out.print("尚无专业");
                    }
                %>
            </td>
                <td width="15%" align="center">
                <input type="hidden" name="<%= "studentid"+i%>" value="<%=rs.getString("studentid") %>">
                <select name="<%= "classid"+i%>">
                <option value="">==尚未分班==</option>
                <%int classid=rs.getInt("classid");
                    sql="select * from classTa";
                    Statement statetemp=conn.createStatement();
                    ResultSet rstemp=statetemp.executeQuery(sql);
                    while(rstemp.next()){

```



```

        int rstempclassid=rstemp.getInt("classid");%>
        <option value="<%=rstempclassid%>"
        <%if(rstempclassid==classid){%> selected<%> %>>
        <%=rstemp.getString("classname")%>
        </option>
        <%}%>
    </select>
</td>
</tr>
<% }%>
<tr>
    <td width="100%" bgcolor="#C0C0C0" align="center" colspan="8">
        <font color="#0000FF">
            <input type="submit" value="确定">
        </font></td>
</tr>
<input type="hidden" name="studentcount" value="<%=i%>">
<input type="hidden" name="action" value="update">
<% } %>
</table>
</form>
</body>
</html>
<%DBConn.closeConn(conn);%>

```

如果要分班操作则 **action** 隐藏域的值为 **update**。在生成表单中的数据时, 提供了如下的数据项:

```

<!--循环体内的提交数据-->
<input type="hidden" name="<%= "studentid"+i%>"
    value="<%=rs.getString("studentid")%>">
<select name="<%= "classid"+i%>">
<option value=">==尚未分班==</option>
<%int classid=rs.getInt("classid");
    sql="select * from classTa";
    Statement statetemp=conn.createStatement();
    ResultSet rstemp=statetemp.executeQuery(sql);
    while(rstemp.next()){
        int rstempclassid=rstemp.getInt("classid");%>
        <option value="<%=rstempclassid%>"
        <%if(rstempclassid==classid){%> selected<%> %>>
        <%=rstemp.getString("classname")%>
        </option>
        <%}%>
    </select>
<!--循环体外的提交数据-->
<input type="hidden" name="studentcount" value="<%=i%>">
<input type="hidden" name="action" value="update">

```

**i** 是一个循环变量。查询出结果后, 在显示每一位学生的数据时使用了 **while** 循环输出数据, 每作一次循环, **i** 变量的值都会增 1。因此在给学生的 ID 号隐藏域起名时, 值为 "**<%= "studentid"+i%>**"; 相应的班级下拉框的名称为 "**<%= "classid"+i%>**"。

在循环体外, 利用一个名为 **studentcount** 的隐藏域提交的要分班的记录条数。在接收数据

后就根据这个记录条数来逐条更新分班情况数据，程序代码如下：

```
String studentcount=request.getParameter("studentcount");
for(int i=1;i<=Integer.parseInt(studentcount);i++){
    String studentid=request.getParameter("studentid"+i);
    String classid=request.getParameter("classid"+i);
    if(classid!=null&&classid.length()!=0&&studentid!=
    null&&studentid.length()!=0){
        String sqlstr="update student set classid="+classid+
        " where studentid="+studentid;
        Statement state=conn.createStatement();
        state.executeUpdate(sqlstr);
    }
}
```

利用“request.getParameter("studentid"+i)”方法可以方便地获得提交的学生 ID 号，再利用同样的方法“request.getParameter("classid"+i)”也可以方便地获得班级的 ID 号，再有针对性地做更新数据处理。

分班情况查询功能则相对比较简单，与学生报到情况查询功能实现方法类似，只是查询条件换成了班级，即按班级查询。此外，收费登记情况查询、宿舍分配情况查询功能所调用的 JSP 页面与分班情况查询功能的相同，均是按班级查询报到中的情况。读者可自行查看随书光盘中的源代码。

## 12.4.8 收费情况登记功能的实现

收费情况登记功能的操作界面如图 12-29 所示。单击左边的“收费情况登记”超链接即可进入到收费情况登记功能操作界面。

序号	姓名	录取通知书号	录取专业	所在班级	交费金额	是否交清
1	陈小明	2008090001	物流管理	物信0902	3900.0	<input checked="" type="radio"/> 是 <input type="radio"/> 否
2	孙东东	2008095342	物流信息	物信0902	400.0	<input checked="" type="radio"/> 是 <input type="radio"/> 否
3	黄婧	2008050643	物流信息	物信0901	0.0	<input type="radio"/> 是 <input checked="" type="radio"/> 否
4	邓佳容	2008070543	物流管理	物信0901	1000.0	<input checked="" type="radio"/> 是 <input type="radio"/> 否
5	陈谈心	30060400004	物流管理	物管0902	100.0	<input type="radio"/> 是 <input checked="" type="radio"/> 否
6	耶和華	2008050601	物流管理	物信0901	100.0	<input checked="" type="radio"/> 是 <input type="radio"/> 否
7	刘是地	2007060505	物流管理	物信0901	300.0	<input checked="" type="radio"/> 是 <input type="radio"/> 否
8	刘小生	2008090008	物流管理	物信0902	3900.0	<input checked="" type="radio"/> 是 <input type="radio"/> 否
9	贺青春	2008090011	物流管理	尚未分班		<input checked="" type="radio"/> 是 <input type="radio"/> 否
10	陈道军	2008090231	物流工程	物信0901	0.0	<input type="radio"/> 是 <input checked="" type="radio"/> 否

图 12-29 收费情况登记功能操作界面

在查询条件表格中输入要查询的学生姓名、录取通知书号，单击“提交”按钮，即可将查



询的结果列在下面的表格中。针对学生报到的情况，将会出现交费情况登录的录入框。

如果还没有分班，则不能交纳学费，如果已经分班，则可录入交费金额，设置学费是否交清单选按钮。来看页面的源代码。

#### acceptmoney.jsp

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ page import="csai.db.DBConn, java.sql.*" %>
<%Connection conn=DBConn.createDBConn(); %>
<%
    //-----获取请求参数值-----
    String studentname=request.getParameter("studentname")+"";
    if(studentname!=null&&studentname.length()!=0)
        studentname=new String(studentname.getBytes("ISO-8859-1"));
    String action=request.getParameter("action")+"";
    String matrino=request.getParameter("matrino")+"";
    //-----构造查询的 SQL 语句-----
    String sqlwhere=new String("");
    String sql=new String("");
    if("select".equals(action)){//如果是查询操作
        if(studentname!=null&&studentname.trim().length()!=0)
            sqlwhere="where studentname like '%" + studentname.trim() + "%' ";
        if(sqlwhere!=null&&sqlwhere.length()!=0){
            if(matrino!=null&&matrino.trim().length()!=0)
                sqlwhere+=" and matrino like '%" + matrino.trim() + "%' ";
        }else{
            if(matrino!=null&&matrino.trim().length()!=0)
                sqlwhere=" where matrino like '%" + matrino.trim() + "%' ";
        }
        sql="select * from student " + sqlwhere;
    }
    //-----交费操作-----
    if("update".equals(action)){//如果是交费操作
        String studentcount=request.getParameter("studentcount");
        for(int i=1;i<=Integer.parseInt(studentcount);i++){
            String studentid=request.getParameter("studentid"+i);
            String payamount=request.getParameter("payamount"+i)+"";
            String payok=request.getParameter("payok"+i)+"";
            if(studentid!=null&&studentid.length()!=0
                &&payamount!=null&&payamount.length()!=0
                &&payok!=null&&payok.length()!=0){
                String sqlstr="update student set payamount="+payamount+", "+
                    "payok="+payok+" where studentid="+studentid;
                Statement state=conn.createStatement();
                state.executeUpdate(sqlstr);
            }
        }
        out.print("收费情况登记成功!");
    }
%>
<html>
<body>
<form method="post" action="acceptmoney.jsp">
```



```

<table border="1" cellpadding="0" cellspacing="0" style="border-collapse:collapse"
    bordercolor="#C0C0C0" width="700">
    <tr>
        <td width="100%" bgcolor="#C0C0C0">
            <font color="#0000FF">要查询的条件</font></td>
        </tr>
    <tr>
        <td width="100%">
            请输入姓名:
            <input type="text" name="studentname">
            请输入录取通知书号:
            <input type="text" name="matrino">
            <input type="hidden" name="action" value="select">
            <input type="submit" value="提交">
        </td>
    </tr>
</table>
</form>
<form action="acceptmoney.jsp" method="post">
<table border="1" cellpadding="0" cellspacing="0" style="border-collapse:collapse"
    bordercolor="#C0C0C0" width="700">
    <tr>
        <td width="100%" bgcolor="#C0C0C0" align="center" colspan="7">
            <font color="#0000FF">查询到的学生数据</font></td>
        </tr>
    <tr>
        <td width="5%" align="center">序号</td>
        <td width="12%" align="center">姓名</td>
        <td width="16%" align="center">录取通知书号</td>
        <td width="12%" align="center">录取专业</td>
        <td width="15%" align="center">所在班级</td>
        <td width="15%" align="center">交费金额</td>
        <td width="15%" align="center">是否交清</td>
    </tr>
    <%if("select".equals(action)){%>
    <%    Statement state=conn.createStatement();
        ResultSet rs=state.executeQuery(sql);
        int i=0;
        while(rs.next()){
            i++;
    <%>
    <tr>
        <td width="5%" align="center"><%=i%></td>
        <td width="12%" align="center"><%=rs.getString("studentname") %></td>
        <td width="16%" align="center"><%=rs.getString("matrino") %></td>
        <td width="12%" align="center">
            <%int specialityid=rs.getInt("specialityid");
            if(specialityid==0)
                out.print("尚无专业");
            else{
                sql="select * from speciality where specialityid="+specialityid;
                Statement statetemp=conn.createStatement();
                ResultSet rstemp=statetemp.executeQuery(sql);
            }
        </td>
    </tr>
    <%>
    </table>
</form>

```



```

        if(rstemp.next())
            out.print(rstemp.getString("specialityname"));
        else
            out.print("尚无专业");
    }
    %>
</td>
<td width="15%" align="center">
<input type="hidden" name="<%= "studentid"+i%>"
        value="<%=rs.getString("studentid")%>"
    <%int classid=rs.getInt("classid");
        boolean isclass=false; //是否已经报到分班?
        sql="select * from classTa where classid="+classid;
        Statement statetemp=conn.createStatement();
        ResultSet rstemp=statetemp.executeQuery(sql);
        if(rstemp.next()){
            out.print(rstemp.getString("classname"));
            isclass=true;
        }else{
            out.print("尚未分班");
        }
    %>
</td>
<td width="15%" align="center">
<%if(isclass){ //如果已经分班%>
<input type="text" name="<%= "payamount"+i%>"
        value="<%=rs.getFloat("payamount")%>" size="12">
<%} %>
</td>
<td width="15%" align="center">
<%if(isclass){ //如果已经分班%>
<input type="radio" name="<%= "payok"+i%>" value="1"
    <%if(rs.getInt("payok")==1) out.print("checked=true"); %>>是
<input type="radio" name="<%= "payok"+i%>" value="0"
    <%if(rs.getInt("payok")==0) out.print("checked=true"); %>>否
<%} %>
</td>
</tr>
<% }%>
<tr>
<td width="100%" bgcolor="#C0C0C0" align="center" colspan="8">
<font color="#0000FF">
<input type="submit" value="确定">
</font></td>
</tr>
<input type="hidden" name="studentcount" value="<%=i%>">
<input type="hidden" name="action" value="update">
<%} %>
</table>
</form>
</body>
</html>
<%DBConn.closeConn(conn); %>

```



如何根据查询的结果动态地在表单中出现文件输入框、单选按钮,提交后又能正确地接收数据并处理,实现的方法与报到分班功能的实现方法相同,此外就不再赘述了,读者可仔细阅读源代码或查看 12.4.7 节中的详细解说。

设置单选按钮被选中的办法是将 radio 标签的 checked 属性值设为 true,此外如果想要使多个单选按钮为同一个组,即同时只能有一个被选中,可将这多项单选按钮的 name 属性值设为相同值。此外,也要根据数据库数据中已有的收费情况来评判,如果已交费的就显示出交费金额和是否交费的单选按钮标志。设置单选按钮标志时使用了如下的程序代码段。

```
<input type="radio" name="<%= "payok"+i%>" value="1"
<%if(rs.getInt("payok")==1) out.print("checked=true"); %>>是
<input type="radio" name="<%= "payok"+i%>" value="0"
<%if(rs.getInt("payok")==0) out.print("checked=true"); %>>否
```

某条记录中的是、否单选钮的名称设为相同,则某一时刻只能选中其中的一个单选按钮。如果从数据库中取出的数据来看, payok 字段值为 1 则表示费用已交清,会设置“是”前的单选按钮被选中。同理,如果 payok 字段值为 0 则表示费用未交清,会设置“否”前的单选按钮被选中。

### 12.4.9 宿舍分配功能的实现

宿舍分配功能的操作界面如图 12-30 所示。单击左边的“宿舍分配”超链接即可进入宿舍分配功能的操作界面。上面的表格用于输入查询的条件,下面的表格显示了查询的结果。

学生报到管理系统 (V1.0)

左侧菜单:

- 报到分班管理
  - 报到分班
  - 分班情况查询
- 报到收费管理
  - 收费情况登记
  - 收费情况查询
- 学生宿舍管理
  - 宿舍分配
  - 宿舍情况查询
- 基础数据管理
  - 录入专业
  - 录入录取学生名册
  - 录入宿舍
  - 录入班级
- 学生报到状况查询
- 用户管理

搜索条件:

请输入姓名:  请输入录取通知书号:

查询到的学生数据:

序号	姓名	录取通知书号	录取专业	所在班级	交费金额	是否交费	所在宿舍
1	陈小明	2008090001	物流管理	物信0902	3900.0	是	学生公寓第2栋
2	孙东东	2008095342	物流信息	物信0902	400.0	是	学生公寓第2栋
3	曹楠	2008050643	物流信息	物信0901	0.0	否	
4	邓佳容	2008070543	物流管理	物信0901	1000.0	是	学生公寓第2栋
5	陈谈心	30060400004	物流管理	物管0902	100.0	否	
6	耶和华	2008050601	物流管理	物信0901	100.0	是	==请选择宿舍==
7	刘是地	2007060505	物流管理	物信0901	300.0	是	==请选择宿舍==
8	刘小生	2008090008	物流管理	物信0902	3900.0	是	学生公寓第2栋
9	贺青春	2008090011	物流管理	尚未分班			
10	陈道军	2008090231	物流工程	物信0901	0.0	否	

图 12-30 宿舍分配功能操作界面

如果学生还没有分班或没有交清学费都是不能设置所在宿舍的,这是如何做到的呢?先看显示交费金额时的代码:

```
<td width="15%" align="center">
<%if(isclass){//如果已经分班%>
<%=rs.getFloat("payamount")%>
<%} %>
</td>
```



也就是说，如果已经分班则可显示所交的学费金额。再看“是否交清”列的显示代码：

```
<td width="15%" align="center">
<%if(isclass){//如果已经分班%>
<%if(rs.getInt("payok")==1) out.print("是"); %>
<%if(rs.getInt("payok")==0) out.print("否"); %>
<%} %>
</td>
```

在已经分班的情况下，如果 **payok** 字段值为 1 表示学费已交清，如果 **payok** 字段值为 0 表示学费未交清。再来看所在宿舍字段中的显示代码：

```
<td width="15%" align="center">
<%if(isclass&&rs.getInt("payok")==1){//如果已经分班且已交清学费%>
<select name="<%= "bedchamberid"+i%>">
<option value="">==请选择宿舍==</option>
<%int bedchamberid=rs.getInt("bedchamberid");
    sql="select * from bedchamber";
    Statement statebed=conn.createStatement();
    ResultSet rsbed=statebed.executeQuery(sql);
    while(rsbed.next()){
        int rsbedbedchamberid=rsbed.getInt("bedchamberid");%>
<option value="<%=rsbedbedchamberid%>"
<%if(rsbedbedchamberid==bedchamberid){%> selected<%} %>>
<%=rsbed.getString("bedchambername")%>
</option>
<%}%>
</select>
<%} %>
</td>
```

如果已经分班且已交清学费，则会显示一个宿舍下拉框，供操作人员选择宿舍。下拉框的 **option** 标签根据宿舍表中的记录条数来定，如果正要增加的 **option** 中，宿舍 ID 号正好等于当前记录的宿舍 ID 号，则将此 **option** 标签设为 **selected**，表示被选中。

此处没有列出本页面的完整源代码，读者可自行查看随书光盘中的源代码。

## 12.5 小结

相信读者通过本章的完整系统开发动手操作后，对软件工程的思想会有更深一步的认识，对用 Eclipse 开发 Web 应用的方法也能熟练操作了，对 JSP 知识的综合运用也更加灵活了，这也是本章要达到的目标。接下来将要进入更难一些的内容了——框架技术。

程序员

Java学习群：72030155

好资料应该和你的朋友分享，把这本电子书分享给学Java的朋友，Java群空间：可以联系群主获取更多大型企业内部技术教程。



# 框架技术篇

- ◆ 第 13 章 Struts 2 框架技术
- ◆ 第 14 章 基于 Struts 2 实现报到管理系统
- ◆ 第 15 章 Hibernate 4 持久化技术
- ◆ 第 16 章 基于 Struts 2+Hibernate 4  
实现报到管理系统
- ◆ 第 17 章 Spring 3 框架技术
- ◆ 第 18 章 基于 SSH 实现报到管理系统

# 13

## Struts 2 框架技术

---

Struts 框架技术是 Java Web 开发人员逐步深入应用需要掌握的一种常用的框架技术，掌握这种技术将给实践工程的开发工作带来许多方便，成倍地提交开发效率。比如采用 Struts 框架技术的 Web 应用能够自动保留表单中的输入数据，能够自动进行常见的数据校验工作，能够通过 OGNL 表达式简化 JSP 页面的代码，实现了 MVC 模式使系统具有更好的可维护性并能适应更大的应用场合等。

本章将采用 Struts 的最新版本 Struts 2 来详细讲解框架的基本原理、应用，以及国际化、OGNL、标签、数据校验等相关知识。

### 13.1 Struts 2 框架介绍

要了解 Struts 框架就需要先了解 MVC 模式，Struts 是实现了 MVC（Model-View-Controller，模型-视图-控制器）模式的框架。模式体现的是一种设计思想，是许多前人的经验性总结，而框架体现的是系统程序的架构，是设计思想在程序开发过程中的实现。

#### 13.1.1 MVC 模式

先一起来理解 MVC 中这 3 个字母对应的词语的含义。

（1）模型（Model）：表示系统的业务逻辑，包括数据和业务上的规则、操作。在 MVC 模式中，模型是要实现系统的核心功能，为视图提供数据，供其他部件调用，对系统的功能起到封装的作用，以提高程序代码的可复用程度。

（2）视图（View）：是系统的外观表现，是系统与用户交互的界面。在视图中并不进行业务处理，只是简单的显示和向模型或控制器提交数据、发送处理请求，在得到处理结果后显示出来，这样就将显示与处理这两种类型的功能区别开来，使程序代码具有良好的层次性和可复用性，各自专注于所擅长的功能。

（3）控制器（Controller）：提供对系统处理过程的控制，对用户的输入做出响应，创建并



设置模型中的属性值，对输入的数据作出校验，根据客户端提出的请求，选择合适的模型来处理业务逻辑，再将数据返回给视图。可见，控制器就相当于一个调度中心和数据处理的中转站。

MVC 模式的基本设计思想如图 13-1 所示。

从图 13-1 可以看出。MVC 模式在实现时把“做什么（业务处理）”和“怎么做（业务实体）”进行了分离，使得开发人员的分工可以更细，业务逻辑可以重用。视图则在一般情况下只接受来自于模型的数据并显示给用户，以及将用户输入的数据提交给模型或控制器。

使用 MVC 设计模式来开发系统的优点主要体现在：

（1）低耦合性。视图和模型的分离，实现了数据显示处理和业务处理逻辑的分别实现，互相之间影响程序较低，改变业务处理的逻辑可以不变更数据显示的逻辑，数据表现的不同形式也不会影响到业务逻辑的处理。

（2）高复用性。比如不同表现的客户端可以共用业务处理逻辑代码，一个 Web 应用系统的核心功能用组件完成后，表现的客户端可以用无线浏览器（WAP），也可以用 Web 浏览器（HTTP），这样不局限于客户端的显示，而核心功能部件是一样的。

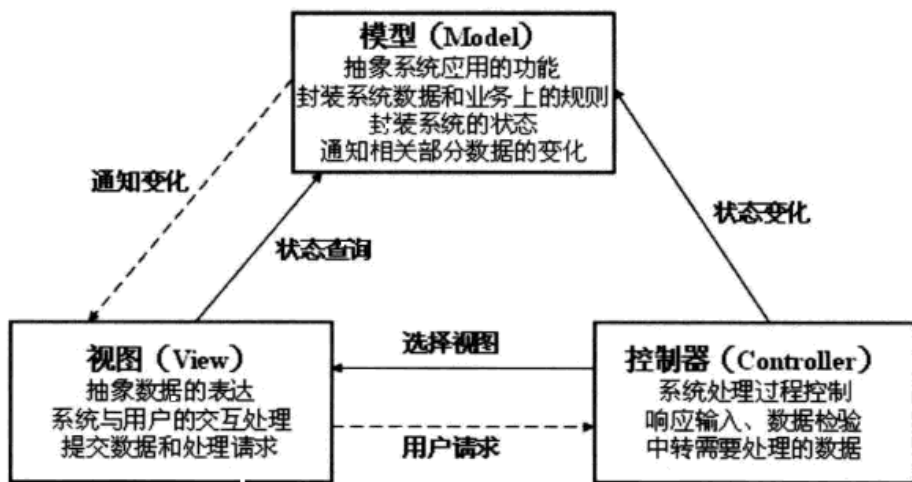


图 13-1 MVC 模式的基本设计思想示意图

### 13.1.2 Struts 2 原理

目前 Struts 有 Struts 1.x（以下简称 Struts 1）和 Struts 2.x（以下简称 Struts 2）这两种主要的版本及其衍生版本在工程实践使用较多，本书中使用 Struts 2，考虑到 Struts 1 将逐步退出历史舞台，本书不再介绍。

Struts 2 在 Struts 1 的基础上作了较大的改进，并融入了 WebWork 2，也实现了 MVC 设计模式。Struts 2 将一个 Web 系统的程序分为模型、视图、控制器三个部分，模型由 JavaBean、EJB 组件等完成具体业务的组件构成；视图由 JSP 文件、POJO 对象（可有可无）组成；控制器由 Action 来实现。

Struts 2 具有如下的特性：

- （1）上手非常容易，可以使用起步教程，使用模板工程或者 Maven 原型建立 Struts 2 工程。
- （2）良好的设计，Struts 2 中想要处理和 HTTP 相关的操作，只需要使用 Struts 框架的接口即

可。在 Struts 2 中不再涉及诸如 `HttpServletRequest`、`HttpServletResponse`、`HttpSession` 等 HTTP 相关的 Servlet 接口类，取而代之的是 Struts 2 的接口，例如 `RequestAware`、`SessionAware` 等。

(3) 标签库功能强大，种类丰富。Struts 2 中所有的标签自带了 Table 内容，可以方便的处理格式。例如 `<s:textfield>` 标签自动添加了 `<tr>`、`<td>` 等标签。有状态的 Checkbox，可以以一种统一的方式记录 checkbox 状态的变化。在 Struts 2 中即使没有被选中的 checkbox，其内容仍然存在于 Struts 2 框架中。

(4) 支持 POJO (Plain Ordinary Java Object / Plain Old Java Object，普通的 JavaBean) 表单。Struts 2 中可以直接使用 Javabean 获得客户的收入或者将属性表示出来，可以使用任意合适的类型来接受页面传来的数据或者将数据表现出来。

(5) 支持众多的其他开源组件和技术扩展。Struts 2 可以方便的使用 Spring 管理 Struts 2 的 action 的创建，通过使用 Spring 可以充分的利用 Spring 的依赖功能，并且能够很好地集成其他的框架，例如 Hibernate，iBatis 等。Struts 2 除了支持 JSP 的表现形式，还支持 JasperReports 报表、JFreechart 图标、Action 链、文件下载等。

Struts 2 框架实现 MVC 设计模式的原理图如图 13-2 所示。

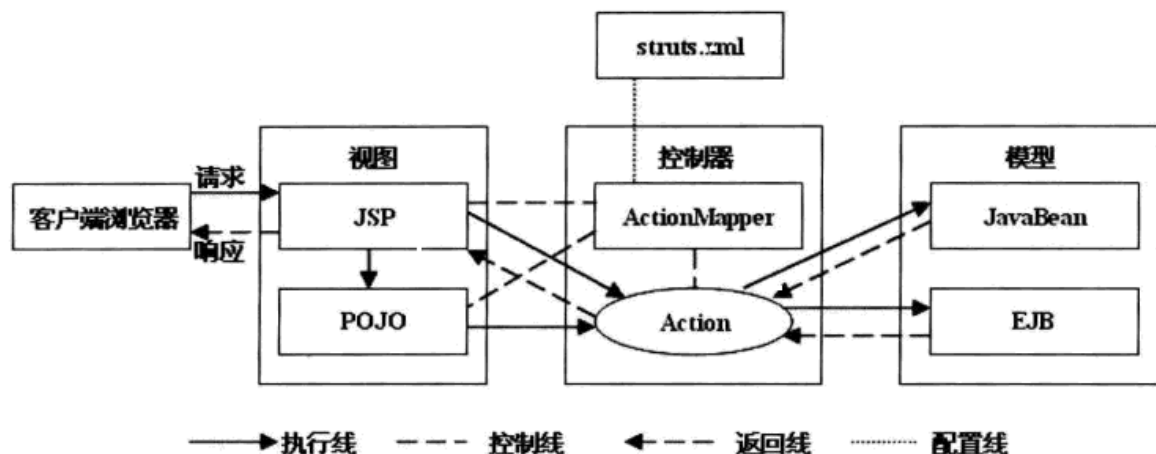


图 13-2 Struts 2 实现 MVC 的原理图

Struts 2 使用 `struts.xml` 文件作为配置文件，`struts.xml` 必须位于 `CLASSPATH` 中，实际工程中常放于当前 Web 应用的“`WEB-INF/classes`”目录中，用户在客户端发出请求后，对应的 POJO 对象、Action 是谁均在此配置文件中作出配置。

**提示** 大多数情况下，为简便起见，在用 Struts 2 开发应用系统时将 POJO 和 Action 合二为一，因此也有文献将 POJO 划分到了控制器中。

一个客户端的请求在 Struts 2 框架中的处理主要有以下几个步骤。

① 客户端初始化一个指向 Servlet 容器（例如 Tomcat）的请求，这个请求经过一系列的过滤器（Filter），接着 `FilterDispatcher` 被调用，`FilterDispatcher` 询问 `ActionMapper` 来决定这个请求是否需要调用某个 Action。其中 `ActionMapper` 在 Web 应用启动时根据配置信息加载生成。

② 如果 `ActionMapper` 决定需要调用某个 Action，`FilterDispatcher` 把请求的处理交给



ActionProxy, ActionProxy 通过 Configuration Manager 询问框架的配置文件, 找到需要调用的 Action 类, ActionProxy 创建一个 ActionInvocation 的实例。ActionInvocation 实例使用命名模式来调用, 在调用 Action 的过程前后, 涉及相关拦截器 (Interceptor) 的调用。

③ 一旦 Action 执行完毕, ActionInvocation 负责根据 struts.xml 中的配置找到对应的返回结果。返回结果通常是 (但不总是, 也可能是另外的一个 Action 链) 一个需要被表示的 JSP。在表示的过程中可以使用 Struts 2 框架中的标签。



**提示** POJO 对象仅是一个拥有若干个属性及属性对应 getXxxx()、setXxxx() 方法的对象, 并不做数据校验, 如果要做数据校验可以在 Action 的 validate() 方法中完成。



**提示** 读者学习了 MVC 模式和 Struts 原理可能觉得还是比较抽象, 那么请注意在后续的内容中一步一步调试程序逐步加深认识, 再回过头来参见原理就会理解得比较深刻了。

### 13.1.3 安装与配置 Struts

从如下的网址可以得到 Struts 的下载列表: <http://struts.apache.org/download.cgi>, 通过浏览器访问这个网址, 如图 13-3 所示。

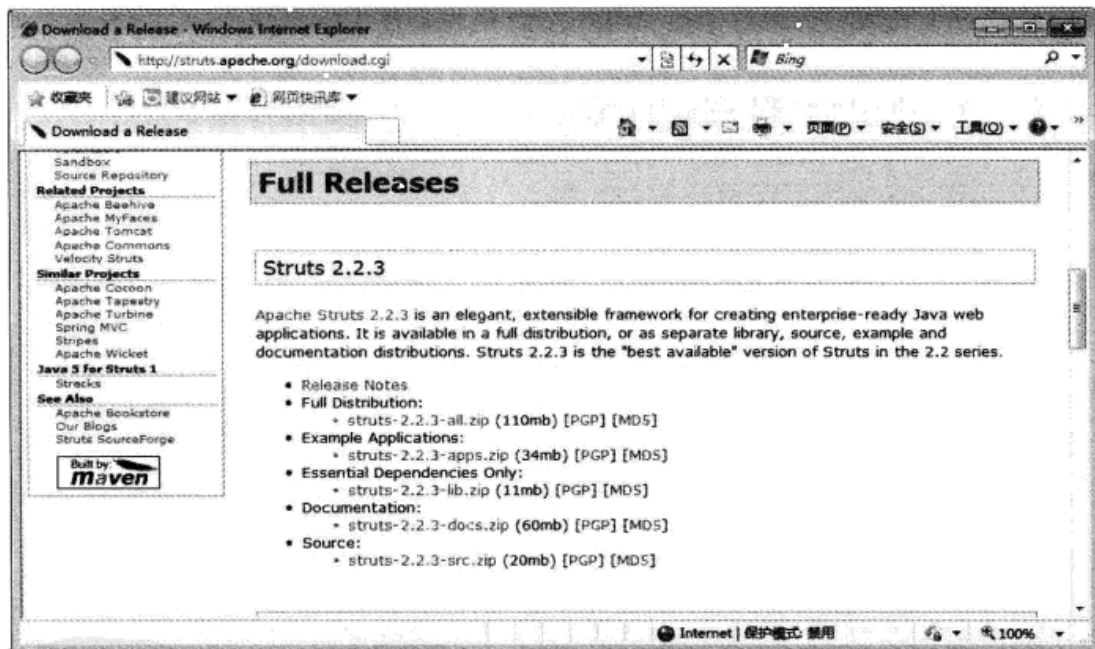


图 13-3 Struts 的下载页面

为方便得到 Struts 的相关资料, 不妨下载 “Full Distribution” 版本, 单击 “Struts 2.2.3” 下的 struts-2.2.3-all.zip 超链接后, 即可下载了, 链接的目标下载地址为:

<http://apache.etoak.com//struts/binaries/struts-2.2.3-all.zip>



**提示** 读者访问此页面时，可能显示的界面会稍有不同，读者可自行下载最新的版本。

下载完毕后得到一个 ZIP 压缩文件，解压缩即可得到 Struts 的所有相关资料，包括 jar 组件包，API 手册，示例代码等。文件包中有 4 个文件夹，各文件夹的内容详见表 13-1。

在 Eclipse 中要使用 Struts 需要把下载得到的文件包中的 lib 子目录下相关的 jar 文件导入工程中，具体如何操作将在后续示例中详细解说。

表 13-1 Struts 2 压缩包中文件夹的内容

文件夹名称	文件夹内容
apps	5 个 war 包示例应用，附带源码，可供读者阅读源码学习
docs	javadoc 和在线文档的离线版本
lib	Struts2 的全部核心类库和依赖包
src	源代码



**提示** 解压 war 可使用“jar、xvf、war”文件名命令。

## 13.2 应用 Struts 2

本节中通过一个“用户登录”的实例并不断改进，来逐步学习如何应用 Struts 2。

### 13.2.1 用 Struts 2 实现用户登录功能

#### 【实例 13-1】用 Struts 2 实现用户登录功能

本例中为实现用户登录的功能，设计的程序流程如图 13-4 所示。

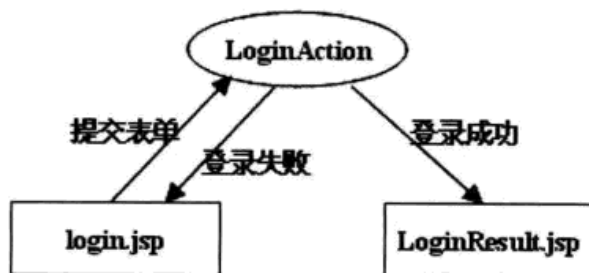


图 13-4 用户登录的程序流程

用户在 login.jsp 页面的表单中输入用户名、密码，再单击“提交”按钮，根据表单的 action 属性值，将表单数据提交到 LoginAction 这一 action 组件；在 LoginAction 中做用户检验，如果检验通过（用户名为“admin”，密码为“pass”），则跳转到 LoginResult.jsp，显示登录的用户名并报告登录成功；如果检验失败就回转到 login.jsp，显示用户名并报告登录失败。



在 Eclipse 中开发 Struts 应用是十分方便的。首先在 Eclipse 中新建一个工程,工程名为 ch13,新建工程即会自动生成 src、build、WebContent 目录(采用默认的设置)。接下来再做一些配置。

### ① 导入 jar 组件包

将 Struts 2 软件包 lib 子目录中的 commons-logging-1.1.1.jar、commons-logging-api-1.1.jar、freemarker-2.3.16.jar、ognl-3.0.1.jar、struts2-core-2.2.3.jar、xwork-2.2.3.jar、common-fileupload-1.2.2.jar、commons-io-2.0.1.jar、commons-lang-2.5.jar、javassist-3.11.0.GA.jar 这 10 个组件包,复制到 WebContent (即当前 Web 应用所在的根目录)的“WEB-INF/lib”目录中。这 10 个 jar 包的说明如表 13-2 所示。

表 13-2 Struts 2 的 10 个重要 jar 包的说明

jar 包名称	说 明
struts2-core-2.2.3.jar	Struts 2 的核心包
xwork-2.2.3.jar	Xwork 2 库, Struts 2 核心包将其作为底层库存在
ognl-3.0.1.jar	Object Graph Navigation Language (OGNL)
freemarker-2.3.16.jar	Struts 2 所有的 ui 标记的模板均使用 freemarker 编写,可通过修改或重写模板使 Struts 2 的 ui 标记按用户的需要进行显示
commons-logging-1.1.1.jar commons-logging-api-1.1.jar	Apache 的 Commons Logging 包,封装了通用的日志接口,可自动调用 Log4J 或者 JDK 1.4 或者更高版本的 util.logging 日志包
common-fileupload-1.2.2.jar	用于文件上传的组件包
commons-io-2.0.1.jar	提供了一些由 Apache 为 JDK 补充的类和方法
commons-lang-2.5.jar	从它的命名上就可以看出它主要是一些公共的工具集合,比如对字符、数组的操作等
javassist-3.11.0.GA.jar	用于操作字节码的组件包



**提示** 不同版本的 Struts 的 lib 子目录中的各种 jar 组件包版本可能会有所差异,这并不影响我们配置程序,在导入组件包时,请读者保证 jar 组件名称中的字母相同。

OGNL (Object Graph Navigation Language) 是一种类似于 EL 的功能强大的表达式语言。将这些 jar 文件放入到“WEB-INF”目录下后,如图 13-5 所示。

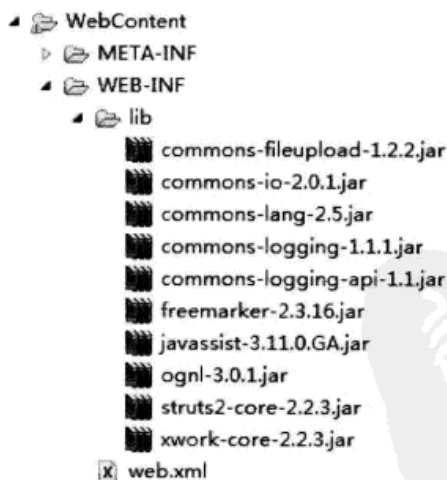


图 13-5 需要放入的 jar 文件

## ② 配置 web.xml

web.xml 位于 Web 应用 WEB-INF 目录下，用于配置 Web 应用，内容如下。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">
  <display-name>ch13</display-name>
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
    </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

该配置文件中，<filter>标签之前的内容都是 Eclipse 自动生成的。文件中还配置了 FilterDispatcher。<display-name>是 Web 应用配置的显示名称，是可选的，可有可无。<filter>中的<filter-name>指出了过滤器的名称，<filter-class>指出过滤器对应的类。<filter-mapping>中的<url-pattern>指出过滤匹配的 URL 模式，设为/\*表示均匹配，这样所有的请求都会过滤，从而通过 FilterDispatcher 查找 actionMapper 的设置来决定请求对应的是哪个 Action。<welcome-file-list>的<welcome-file>指出进入 Web 应用访问的默认的文件。



**提示** 这里的 web.xml 中的配置是相对通用的，读者在工程实践中可以复制这个文件中的内容，再根据需要适当修改即可。

## ③ 编写 JSP 文件

根据程序流程的设计，需要编写两个 JSP 文件：login.jsp 和 loginResult.jsp，这两个文件的源代码分别如下。

login.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head><title>登录系统</title></head>
<body><br><br><br><br>
<div align="center">
```



```

${requestScope.message}
<s:form action="Login" method="POST">
    <s:textfield name="adminUserName" label="用户名"/>
    <s:password name="adminUserPassword" size="21" label="密码"/>
    <s:submit value="提 交"/>
</s:form>
</div>
</body>
</html>

```

由于文件中要用到 Struts 的标签，故文件首部使用了如下语句：

```
<%@ taglib prefix="s" uri="/struts-tags"%>
```

这样，在使用 Struts 2 的标签时就需要在标签名称前加入“s:”。

“\${requestScope.message}”是一个 EL 表达式，表示显示 request 范围内的变量 message 的值。这里的 message 指的是在 LoginAction 中用户登录校验失败返回的报错信息，在 Action 中将 message 设为一个属性，用户登录校验失败后，设置 message 属性的值，系统即会将 message 属性的值保存在 request 范围中。

声明一个表单使用<s:form>标签，action 属性值为 Login，Login 是 Struts 配置文件中指出的 action 配置的名称。<s:textfield>标签表示文本输入框，<s:password>标签表示密码输入框；<s:submit>标签表示提交按钮。

在 Struts 2 中，<s:form>、<s:textfield>、<s:password>、<s:submit>这些表单标签会自动生成<table>、<tr>、<td>，login.jsp 最终回传给用户浏览器的代码如下。

#### login.jsp

```

<html>
<head><title>登录系统</title></head>
<body><br><br><br><br>
<div align="center">
<form id="Login" name="Login" onsubmit="return true;" action="/ch13/Login.action"
    method="POST">
<table class="wwFormTable">
    <tr>
    <td class="tdLabel">
<label for="Login_adminUserName" class="label">用户名:</label></td>
    <td>
<input type="text" name="adminUserName" value=""
    id="Login_adminUserName"/></td>
</tr>
    <tr>
    <td class="tdLabel">
<label for="Login_adminUserPassword" class="label">密码:</label></td>
    <td><input type="password" name="adminUserPassword" size="21"
    id="Login_adminUserPassword"/></td>
</tr>
    <tr>
    <td colspan="2">
<div align="right"><input type="submit" id="Login_0" value="提 交"/>
</div></td>
</tr>

```

```

</table>
</form>
</div>
</body>
</html>

```

对比即可知，系统自动生成了<form>的 id、name 属性值，值为目标 Action 的名称；<form>的 action 属性值为“/ch13/Login.action”，原来<s:form>的 action 属性为“Login”，系统自动做了转换。为表单中的输入控件自动生成了 id 属性值，如果控件有 name 属性值则自动生成的 id 属性值生成规律为“表单 id 属性值\_控件 name 属性值”，如果没有 name 属性值则自动生成的 id 属性值生成规律为“表单 id 属性值\_数字 n”，数据 n 的值从 0 开始。

#### loginResult.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<html>
<head>
<title>验证通过</title>
</head>
<body>
  ${message}
</body>
</html>

```

这个文件的功能仅仅只是显示验证结果 message 的值而已。

#### ④ 配置 struts.xml

将 struts.xml 放置于“WEB-INF/classes”目录下。在 Eclipse 中开发时，只需要在“Java Resources:src”中新建 struts.xml 文件，则会自动在 build 目录中生成 struts.xml，即实际运行时 Web 应用的“WEB-INF/classes”目录。struts.xml 文件的内容如下。

#### struts.xml

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="example" namespace="" extends="struts-default">
    <action name="Login" class="action.LoginAction">
      <result>/loginResult.jsp</result>
      <result name="input">/login.jsp</result>
    </action>
  </package>
</struts>

```

<struts>与</struts>之间的内容是 actionMapper 相关的配置。一个 Web 应用系统可能会有相当多的配置，因此需要使用<package>分门别类地进行整理，这种组织方式就好像是 Java 类开发时使用的包。



**提示** struts.xml 文件的配置还会在本章后续内容中作出详细解释。



<action>的 class 属性指出了 Action 对应的类, <result>指出 Action 执行完毕后要导向的目标页面。<result>的 name 表示 Action 的 execute()方法返回的相应字符串。如果没有指出 name 属性值, 则默认为“success”。

### ⑤ 编写 Action

LoginAction.java 的源代码如下。

LoginAction.java

```
package action;
import com.opensymphony.xwork2.ActionSupport;
/**
 * 用户登录功能的 Action 类
 * @author dengziyun
 *
 */
public class LoginAction extends ActionSupport {
    /**
     * 执行用户验证的方法
     */
    public String execute() throws Exception {
        if("admin".equals(adminUserName)&&"pass".equals(adminUserPassword))
            message=adminUserName+"登录成功! ";
        else{
            message=adminUserName+"登录失败! ";
            return INPUT;
        }
        return SUCCESS;
    }
    public String adminUserName;//用户名
    public String adminUserPassword;//密码
    public String message;//execute()执行完后返回的消息
    /**
     * 此处省去 adminUserName、adminUserPassword、message 三个属性的 getXxxx() 方法
     * 和 setXxxx() 方法代码
     */
}
```

这个类有三个属性: adminUserName 属性、adminUserPassword 属性与 login.jsp 页面中的用户登录表单输入项名称一一对应, message 属性表示执行完 execute()后返回的消息。

execute() 方法有两种返回值: INPUT 或 SUCCESS。SUCCESS 和 INPUT 都在 com.opensymphony.xwork2.Action 接口中进行了定义, 类型为 String, 分别用来表示输入 (“input”) 和成功 (“success”)。

com.opensymphony.xwork2.ActionSupport 类实现了 com.opensymphony.xwork2.Action 接口, 而 LoginAction 又继承了 com.opensymphony.xwork2.ActionSupport 类, 故 SUCCESS 和 INPUT 这是个常量可以在 LoginAction 中直接使用。

运行程序的第一个页面为 login.jsp, 执行结果如图 13-6 所示。

图 13-6 所示的输入将会验证失败, 失败后又会跳转回 login.jsp, 显示结果如图 13-7 所示。

从图 13-7 中可以看出, 登录失败时会将错误信息显示在表单上方。前面输入的用户名在

失败后导向的目标页面中仍然保留了，这样减少了程序员的负担，不必再编程实现保留之前已经输入过的数据。但密码输入框不属于保留数据的控件类型之中。



图 13-6 login.jsp 页面



图 13-7 登录失败时的情形

### 13.2.2 解决中文乱码的问题

在实例 13-1 中，当输入的用户名和密码是英文时，操作和显示都没有问题；但当输入中文时就会出现乱码的情况。为解决 Struts 2 开发过程中出现的各种中文字符乱码的问题，读者可以在程序中作出如下的变更。

#### 1. JSP 首部语句

将每个 JSP 页面首部的如下语句：

```
<%@ page contentType="text/html; charset=gb2312" %>
```

修改为：

```
<%@ page contentType="text/html; charset=utf-8" %>
```

也就是说将字符编码设置为“utf-8”。



## 2. struts.properties

在当前 Web 应用的“WEB-INF/classes”目录中增加一个 struts.properties，内容如下（如有这两项配置就修改）：

```
struts.locale=zh_CN
struts.i18n.encoding=UTF-8
```

struts.properties 文件定义了 Struts 2 框架的大量属性，读者可以通过改变这些属性来满足应用的需求。struts.properties 文件是一个标准的 Properties 文件，该文件包含了系列的 key-value 对象，每个 key 就是一个 Struts 2 属性，该 key 对应的 value 就是一个 Struts 2 属性值。

struts.properties 文件通常放在 Web 应用的“WEB-INF/classes”路径下。实际上，只要将该文件放在 Web 应用的 CLASSPATH 路径下，Struts 2 框架就可以加载该文件。

此外，struts.properties 文件的内容均可在 struts.xml 中以如下的形式来加入配置：

```
<constant name="" value=""></constant>
```

## 13.3 国际化应用程序

其实在 Struts 1 中就支持国际化的消息显示，例如当网站分别被运行在中英文环境下的浏览器访问时，会使用不同的资源文件来显示中文或者英文的界面，但是那时候的支持相对来说比较麻烦。在 Struts 1 中，如果您要输出一条国际化的信息，只需在代码包中加入“文件名\_xx\_XX.properties”，然后在 struts-config.xml 中指明其路径，再在页面中用<bean:message>标示输出即可。Struts 2 在原有的简单易用的基础上，将其做得更灵活、更强大，国际化的消息支持要相对 Struts 1 简单得多。

### 13.3.1 为用户登录功能加入国际化处理

#### 【实例 13-2】为用户登录功能加入国际化处理

下面仍以实例 13-1 为例来讲解。比如要在登录页面的上方显示一个国际化的欢迎语句，中文为“欢迎您！”，英文为“Welcome you!”，来看如何进行改造。

##### ① 修改 struts.properties

在 struts.properties 文件中加入如下的配置项：

```
struts.custom.i18n.resources=message
```

message 是资源文件的名称，根据语言的不同，分别位于不同的 properties 文件中，比如中文位于 messages\_zh\_CN.properties 文件中，英文则位于 messages\_en\_US.properties 文件中，均需要做转码处理，转码处理使用 native2ascii.exe 工具。



**提示** 如果要加载多个 properties 文件做国际化处理，则不同的文件名之间用逗号（,）隔开。

## ② 编辑 message.properties 及做编码转换

native2ascii.exe 工具位于 JDK 的安装目录的 bin 目录中。查看系统变量 path 中是否设置了 JDK 安装目录的 bin 目录, 如果没有则将其添加上。或在使用 native2ascii.exe 工具前执行如下命令:

```
set path=%path%;%JAVA_HOME%/bin;
```



**提示** 执行这个命令要与使用 native2ascii.exe 工具的命令在同一个命令会话(即打开的同一个 command 窗口)中。

用 native2ascii.exe 工具执行如下命令:

```
native2ascii -encoding gb2312 源.properties 文件名 目标.properties 文件名
```

如要把当前目录下的 message.properties 文件进行转码, 转码后的文件名为 message\_message\_zh\_CN.properties, 其命令为:

```
native2ascii -encoding gb2312 message.properties message_zh_CN.properties
```

在当前 Web 应用中的 src 目录新建一个 message.properties 文件, 内容如下:

```
welcome.message=欢迎您!
```

新建一个 message\_en\_US.properties 文件, 内容如下:

```
welcome.message=Welcome you!
```

在命令窗口将 message.properties 文件转码成 message\_zh\_CN.properties 文件的过程如图 13-8 所示。

可见过程是先用 cd 命令将当前目录切换到 message.properties 文件所在的目录, 再用 native2ascii 工具进行转换。

## ③ 修改 login.jsp

修改 login.jsp, 在其 "\${requestScope.message}" 表达式前加入如下的语句:

```
<s:text name="welcome.message"/>
```

程序运行结果如图 13-9 所示。

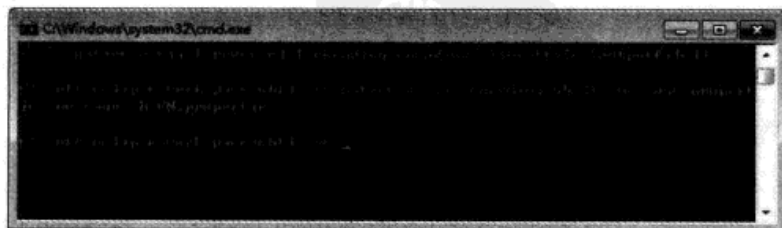


图 13-8 将 message.properties 文件转码成 message\_zh\_CN.properties 文件



图 13-9 加入中文消息



如果将浏览器的默认语言改为“英语（美国）”，刷新页面，将出现英文的“Welcome you!”。

### 13.3.2 查找资源文件的顺序

在 Java 开发中将.properties 文件称为一种资源文件，在 Struts 2 中使用资源文件有如下的 4 种做法：

（1）使用全局的资源文件，方法如实例 13-2 所示。这种方法适用于遍布整个应用程序的国际化字符串，它们在不同的包中被引用，如一些比较共用的出错提示。

（2）使用包范围内的资源文件。做法是在包的根目录下新建名的 package.properties 和 package\_xx\_XX.properties 文件，这就适用于在包中不同类访问的资源。

（3）使用 Action 范围的资源文件。做法为在 Action 的包下新建文件名（除文件扩展名外）与 Action 类名同样的资源文件，则此资源文件只能在该 Action 中访问。这样就可以在不同的 Action 里使用相同的属性名称表示不同的值。例如，在 ActionOne 中 title 为“动作一”，而同样用 title 在 ActionTwo 中表示“动作二”，以提供一定的通用性。

（4）使用<#s:18n>标志访问特定路径的 properties 文件。使用时要注意<#s:18n>标志的范围。在<#s:18n name="xxxxx">到</s:18n>之间，所有的国际化字符串都会在名为 xxxxx 资源文件中查找，如果找不到，Struts 2 就会输出默认值（国际化字符串的名字）。

根据以上 4 种方法，查找的范围从大到小，但实际查询的方法是从小到大，这一点请特别注意。

在 Action 中也可以使用表达式来得到资源消息的值，比如在实例 13-2 的 LoginAction 类中，要得到 welcome.message 的值，可以使用如下的方法：

```
getText("welcome.message")
```

下面来举个例子。假设要在某个 loginAction 类（类名）中调用了 getText("user.message")，Struts 2 将会遵行如下的查找顺序：

- （1）查找 LoginAction\_xx\_XX.properties 文件或 loginAction.properties。
- （2）查找 LoginAction 实现的接口，查找与接口同名的资源文件 MyInterface.properties。
- （3）查找 LoginAction 父类的 properties 文件，文件名为“父类名.properties”。
- （4）判断当前 loginAction 是否实现接口 ModelDriven。如果是，调用 getModel() 获得对象，查找与其同名的资源文件。
- （5）查找当前包下的 package.properties 文件。
- （6）查找当前包的父包，直到最顶层包。
- （7）在值栈（Value Stack）中，查找名为 user 的属性，转到 user 类型同名的资源文件，查找键为 title 的资源。
- （8）查找在 struts.properties 配置的默认的资源文件。
- （9）如果还是没有找到，则输出 user.title。



**提示** 以上过程是自动完成的，并不需要读者去编程实现。

### 13.3.3 参数化字符串

在许多情况下需要在运行时为国际化字符插入一些参数，例如在输入验证提示信息的时候。在 Struts 2 中有以下的两种方法来进行参数化：

(1) 在资源文件的国际化字符串中使用 **OGNL** 表达式，格式为：\${表达式}，例如：

```
validation.password.null=用户名${getText(password)}输入为空
```



**提示** 有关 OGNL 表达式的内容，在本章的后续内容中还有详细介绍。

(2) 使用 `java.text.MessageFormat` 中的字符串格式，格式为：{参数序号（从 0 开始），格式类形（number|date|time|choice），格式样式}，例如：

```
validation.between=日期必须介于{0, date, short}和{1, date, short}之间
```

在显示这些国际化字符串时，同样有两种方法设置参数的值：

(1) 使用标志的 `value0`、`value1...valueN` 的属性，如：

```
<s:text name="validation.password.null" value0="administrator"/>
```

(2) 使用 `param` 子元素，这些 `param` 将按先后顺序，代入到国际化字符串的参数中，例如：

```
<s:text name="validation.password.null">
  <s:param value="administrator"/>
</s:text>
```

## 13.4 OGNL 表达式

在 JSP 页面中使用“<%”和“%>”包括起来的 Java 代码块的方法，会让人感觉页面的流程、程序都比较杂乱，原因是 Java 代码和 HTML 代码夹杂在一起。书写表达式可以让 JSP 页面变得更简洁，而且程序流程相当清晰，代码量也少了许多。

### 13.4.1 Struts 2 对表达式的支持

Struts 2 中支持以下几种表达式语言：

(1) **OGNL**：一种可以方便地操作对象属性的开源表达式语言。

(2) **EL**：JSP 2.0 集成的标准表达式语言。

(3) **Groovy**：基于 Java 平台的动态语言，它具有当前比较流行的动态语言（如 Python、Ruby 和 Smarttalk 等）的一些特性。

(4) **Velocity**：严格来说不是一种表达式语言，它是一种基于 Java 的模板匹配引擎。

Struts 2 默认的表达式语言是 **OGNL**，原因是它相对其他表达式语言具有下面几个优点：

(1) 支持对象方法调用，如：对象名称.方法名称()。这在 EL 表达式中是不可以的，EL 表达式只能通过设置属性值来调用对象的 `getXxx()` 方法和 `setXxx()` 方法。



(2) 支持类的静态方法调用和值访问，表达式的格式为：

```
@[类全名(包括包路径)]@[方法名|值名]
```

例如：

```
//调用 String 类的 format 静态方法
@java.lang.String@format('foo%s', 'bar')
```

或：

```
//得到 tutorial.MyConstant 的静态属性 APP_NAME 的值
@tutorial.MyConstant@APP_NAME;
```

(3) 支持赋值操作和表达式串联，如：

```
price=100, discount=0.8, calculatePrice()
```

这个表达式会返回 80。

(4) 访问 OGNL 上下文和 Action 上下文 ActionContext。

(5) 可以操作集合对象。

13.4.2 使用 OGNL 表达式

在 OGNL 中有一个类型为 Map 的上下文，在这个上下文中有一个根元素（root），对根元素属性的访问可以直接使用属性名字，但是对于其他非根元素属性的访问必须加上特殊符号“#”。

在 Struts 2 中上下文为 Action 上下文（ActionContext），根元素为值堆栈（Value Stack），值堆栈代表了一簇对象而不是一个对象，其中 Action 类的实例也属于值堆栈的一个。OGNL 上下文、Action 上下文等元素的关系如图 13-10 所示。

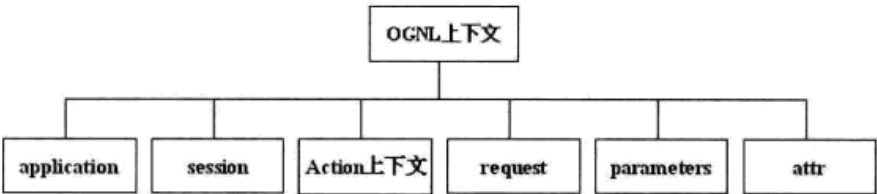


图 13-10 OGNL 上下文、Action 上下文等元素的关系图

访问 OGNL 上下文和 Action 上下文，“#”相当于 ActionContext.getContext()。表 13-3 对 OGNL 上下文中的元素做出了说明。

表 13-3 OGNL 上下文中的元素

名 称	作 用	例 子
parameters	包含当前 HTTP 请求参数的 Map	#parameters.id[0]作用相当于 request.getParameter("id")
request	包含当前 HttpServletRequest 属性的 Map	#request.userName 相当于 request.getAttribute("userName")
session	包含当前 HttpSession 属性的 Map	#session.userName 相当于 session.getAttribute("userName")
application	包含当前应用的 ServletContext 属性的 Map	#application.userName 相当于 application.getAttribute("userName")

续表

名 称	作 用	例 子
attr	用于按 request→session→ application 的顺序访问其属性	#attr.userName 相当于按顺序在以上三个范围内读取 userName 属性, 直到找到为止
Action 上下文	主要用于访问 Action 的属性, 可以不用 #	userName 相当于读取 Action 的 userName 属性



**提示** 初学者可能会对 OGNL 上下文、Action 上下文、值堆栈这些术语不太理解, 请结合以上文字及后续实例逐步体会。

OGNL 表达式常用 `<s:property>` 标签来配合使用。在 JSP 页面中读取 Action 的属性用如下的形式:

```
<s:property value="Action 属性的名称" />
```

OGNL 上下文中非 Action 上下文元素的属性可以按照如下的方式访问:

```
<s:property value="#session.会话范围属性的名称" />
```

或

```
<s:property value="#session["会话范围属性的名称"]" />
```

或

```
<s:property value="#request["请求范围属性的名称"]" />
```

在 Action 类中可以通过 `ActionContext` 中的静态方法来访问 `ActionContext`, 如:

```
//设置会话范围属性 mySessionPropKey 的值
ActionContext.getContext().getSession().put("mySessionPropKey", mySessionObject);
```

### 【实例 13-3】OGNL 表达式使用示例

仍以实例 13-1 为基础, 修改 `loginAction.java` 为 `loginAction1.java`, 在这个 Action 类的 `execute()` 方法中写入一个会话范围的属性; 修改 `struts.xml` 文件中的配置; 修改 `login.jsp` 为 `login1.jsp`, 在其中加入 OGNL 表达式显示一些属性的值。

#### loginAction1.java

```
package action;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
/**
 * 用户登录功能的 Action 类
 * @author dengziyun
 *
 */
public class LoginAction1 extends ActionSupport {
    /**
     * 执行用户验证的方法
     */
    public String execute() throws Exception {
```





```
<s:property value="#session.sessionTestAttr"/>
```

现在来做一个测试，在访问 login.jsp 页面时，输入用户名“test 管理员”，结果如图 13-11 所示。



图 13-11 使用 OGNL 表达式

在 LoginAction1 类中，为方便设置和取得 session、application、request 范围中的变量值，可以将 Action 设为扩展了的 ServletRequestAware、SessionAware、ServletContextAware 接口，比如可以做如下的修改：

#### LoginAction1.java

```
package action;
import java.util.Map;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.interceptor.ServletRequestAware;
import org.apache.struts2.interceptor.SessionAware;
import org.apache.struts2.util.ServletContextAware;
import com.opensymphony.xwork2.ActionSupport;
/**
 * 用户登录功能的 Action 类
 * @author dengziyun
 */
public class LoginAction1 extends ActionSupport
implements ServletRequestAware, SessionAware, ServletContextAware{
    private HttpServletRequest request;
    private Map<String, String> session;
    private ServletContext application;
    public void setServletRequest(HttpServletRequest request) {
        this.request = request;
    }
    public void setSession(Map session) {
        this.session = session;
    }
    public void setServletContext(ServletContext application) {
        this.application = application;
    }
}
```



```

/**
 * 执行用户验证的方法
 */
public String execute() throws Exception {
    session.put("sessionTestAttr", new String("会话测试变量值"));
    if("admin".equals(adminUserName)&&"pass".equals(adminUserPassword))
        message=adminUserName+"登录成功! ";
    else{
        message=adminUserName+"登录失败! ";
        return INPUT;
    }
    return SUCCESS;
}
public String adminUserName;//用户名
public String adminUserPassword;//密码
public String message;//execute()执行完后返回的消息
/**
    此处省去 adminUserName、adminUserPassword、message 三个属性的 getXxxx() 方法
    和 setXxxx() 方法代码
 */
}

```

再调试一次程序将和例子中的效果相同。然而,根据笔者的经验,并不提倡程序员这么做,因为 request 范围的变量值应当设为 Action 的属性, application、session 以及 ServletContext 都可以通过 ActionContext 来获得,而不必编写上述程序中如此多的代码。开发人员可通过如下的语句得到 request、response 对象:

```

ActionContext ctx = ActionContext.getContext();
HttpServletRequest request =
    (HttpServletRequest)ctx.get(ServletActionContext.HTTP_REQUEST);
HttpServletResponse response =
    (HttpServletResponse)ctx.get(ServletActionContext.HTTP_RESPONSE);

```

### 13.4.3 值堆栈

XWork 在 OGNL 之上提供的最大改进就是支持值堆栈, OGNL 允许上下文中保存多个对象。在 XWork 中,整个值堆栈是上下文的根对象,不仅仅只根据表达式从栈中获取对象,还从对象中获取属性, XWork 有一个特殊的 OGNL PropertyAccessor (属性访问器),它可以自动搜索栈内的所有实体(从上到下),直到找到一个具有与你所寻找的对象和属性匹配的。

下面假设值堆栈中包括两个对象: Animal (动物) 和 Person (人),两个对象都有“name”属性, Animal 具有一个“species”属性,而 Person 有一个“salary”属性。Animal 是栈顶元素, Person 在它后面,如图 13-12 所示。

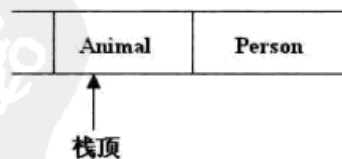


图 13-12 值堆栈中的 Animal 对象和 Person 对象

请参见如下的代码：

```
species    // 调用 animal.getSpecies()
salary     // 调用 person.getSalary()
name       // 调用 animal.getName() 因为 animal 是栈顶元素
```

最后一个 OGNL 表达式将让 Animal 的 name 被返回，一般情况下这就是开发人员所要的效果，根据栈的 FILO (First In Last Out, 先进后出)，也就是说 Animal 在 Person 之前，故默认情况下会先得到 Animal 的 name 属性值。但是有时也可能希望访问的是顺序靠后的对象的属性，为方便处理这种情况，XWork 还提供了按照索引来访问值堆栈的方法。参见如下的代码：

```
[0].name    // 调用 animal.getName()
[1].name    // 调用 person.getName()
```



**提示** 索引是从 0 开始的。

## 13.4.4 OGNL 与集合

### 1. 生成 List

生成 List 的语法为：

```
{e1,e2,e3,...}
```

如下的语句：

```
<s:select label="请选择" name="name"
  list="{ '名称 1', '名称 2', '名称 3' }" value="%{'名称 2'}" />
```

利用 OGNL 表示的 List 对象配合<s:select>标签生成了一个<select>下拉框，默认情况下生成的<select>下拉框代码为：

```
<tr>
  <td class="tdLabel"><label for="name" class="label">请选择:</label></td>
  <td>
    <select name="name" id="name">
      <option value="名称 1">名称 1</option>
      <option value="名称 2" selected="selected">名称 2</option>
      <option value="名称 3">名称 3</option>
    </select></td>
</tr>
```

### 2. 生成 Map

生成 Map 的语法为：

```
#{key1:value1,key2:value2}
```

如下的语句：

```
<s:select label="请选择" name="name2"
  list="#{'1':'名称 1','2':'名称 2'}" value="%{'2'}" />
```

利用 OGNL 表示的 Map 对象配合<s:select>标签生成了一个<select>下拉框，默认情况下生成的<select>下拉框代码为：



```

<tr>
  <td class="tdLabel"><label for="name2" class="label">请选择:</label></td>
  <td>
    <select name="name2" id="name2">
      <option value="1">名称 1</option>
      <option value="2" selected="selected">名称 2</option>
    </select></td>
  </tr>

```

可见 Map 对象的 key 被填充在<option>的 value 属性中,value 被填充在<option>与</option>之间。

### 3. 判断是否存在 List 之中

判断一个对象是否存在 List 中,可使用“in”,一个示例如下:

```

<s:if test="'名称 1' in {'名称 1','名称 2'}">
  名称 1 存在 List 集合中。
</s:if>
<s:else>
  名称 1 不存在 List 集合中。
</s:else>

```

程序代码调试的结果是显示“名称 1 存在 List 集合中。”

如果要判断一个对象是否不存在,则使用“not in”。将上面的示例代码稍作修改,如下所示:

```

<s:if test="'名称 1' not in {'名称 1','名称 2'}">
  名称 1 不存在 List 集合中。
</s:if>
<s:else>
  名称 1 存在 List 集合中。
</s:else>

```

程序调试的结果仍然是显示“名称 1 存在 List 集合中。”

### 4. 取得 List 的一部分

取得 List 的一部分可以使用“?”、“^”、“\$”这三个符号,分别表示的含义如下:

?: 所有满足选择逻辑的对象。

^: 第一个满足选择逻辑的对象。

\$: 最后一个满足选择逻辑的对象。

例如,如下的 OGNL 表达式:

```
person.relative.{? #this.gender == 'male'}
```

上述代码取得这个人(List 对象)所有的男性(this.gender==male)亲戚(relatives)。

## 13.5 Struts 2 标签

Struts 2 标签库中的标签相当丰富,可以帮助开发人员简化 JSP 页面的程序编制工作。Struts 2 的标签分为通用标签和用户界面标签两大类。通用标签主要用于控制页面的执行流程;用户界

面标签主要用于显示数据。

通用标签包括控制标签和数据标签；用户界面标签主要包括表单标签、非表单用户界面标签。



**提示** 后续内容在介绍 Struts 2 标签时，标签的名称都以“<s:”开始，其实“<s:”不是必须的，可以由程序员在 JSP 页面用首部的如下语句来指定标签的前缀：

```
<%@ taglib prefix="s" uri="/struts-tags"%>
```

prefix 属性用于指定 Struts 2 标签的前缀。为描述方便，后续内容使用“<s:”作为前缀。

### 13.5.1 标签属性值的设置

在本书前面章节中学习 JSTL 时，JSTL 中标签的属性值可以用 EL 表达式或字符串表示，在 Struts 2 中标签属性值的设置方法将会更加丰富。

如果标签用于显示动态的数据，比如一个文本输入框，在 Struts 2 中常用如下的语句生成：

```
<s:textfield name="postalCode"/>
```

postalCode 是输入框的名称，表示邮政编码。如果值堆栈中可以找到 postalCode，则输入框 postalCode 的值会被设置为值堆栈中 postalCode 属性的值。这种情况常用于表单提交数据后，在 Action 中作出处理，如果处理后仍返回表单数据输入界面，则 Action 的属性将保存在值堆栈中，从而可以在输入表单中自动设置输入项的值，起到保留输入值的作用。

如果需要动态设置标签的属性值，可以使用表达式，比如设置为当前 Web 应用的资源文件（.properties 文件）中的值，参见如下的语句：

```
<s:textfield key="postalCode.label" name="postalCode"/>
```

此语句中将输入框前的提示文件值设为消息资源中的 postalCode.lable 值。

有些标签的属性值数据类型并非字符串，如 boolean、int 等。Struts 2 会把属性中的设置值作为表达式计算，并将结果自动转换为所需的数据类型。参见如下的语句：

```
<s:select key="state.label" name="state" multiple="true"/>
```

multiple 属性值的数据类型为 boolean，Struts 2 会将 true 作为一个布尔值，而不是一个字符串。

表达式虽然能够自动求值，但有时也可能会带来一些麻烦，比如：

```
<s:textfield key="state.label" name="state" value="ca"/>
```

Struts 2 会自动用 getCa() 方法到值堆栈中去找是否有 ca 这个属性，再把这个属性值找出来，但是这并不一定是开发人员想要的结果，有可能开发人员只是想简单地将标签的值设为字符串“ca”，这时就可以使用“%{...}”符号了，可以将如下的语句修改为：

```
<s:textfield key="state.label" name="state" value="%{'ca'}"/>
```

标签的属性值求值遵循如下的规则：

(1) 所有的字符串属性使用“%{...}”符号解析。



(2) 所有非字符串属性不解析，而直接作为表达式求值。

(3) 如果第 2 种情况是因为有“%{...}”符号而产生异常，则相当于去掉“%{...}”符号，再作为表达式求值。

“%”符号的用途是在标签的属性值为字符串类型时，计算 OGNL 表达式的值。



**提示** 默认情况下使用 Struts 2 的标签时会自动生成 HTML 的表格标签，如果不需要生成，可将标签的 theme 属性设为 simple，或在 struts.properties 文件中增加如下的配置：

```
struts.ui.theme=simple//设置不自动生成表格标签
```

## 13.5.2 控制标签

### 1. <s:if>标签、<s:elseif>标签与<s:else>标签

<s:if>标签用于条件判断，可以单独使用，或结合一个或多个<s:elseif>标签一起使用，或结合一个<s:else>标签一起使用。<s:if>标签的使用形式如下：

```
<s:if test="条件表达式 1">
    条件表达式为 true 时执行的代码
</s:if>
[<s:elseif test="条件表达式 2">
    如果条件表达式 1 为 false，且条件表达式 2 为 true 时执行的代码
</s:elseif>
... ..]
[<s:else>
    以上条件表达式都为 false 时执行的代码
</s:else>
]
```

<s:if>标签和<s:elseif>标签有一个属性 test，数据类型为 boolean，这个属性中的表达式决定是否执行标签间的代码，如果表达式成立则执行，否则不执行。

### 2. <s:iterator>标签

<s:iterator>标签用于迭代处理 java.util.Collection 对象或 java.util.Iterator 对象。使用<s:iterator>标签的形式如下：

```
<s:iterator [status="状态变量名称"] value="代表迭代的当前元素的变量名称"
    var="放入值堆栈中的变量名称">
```

迭代过程中的语句：

```
</s:iterator>
```

<s:iterator>标签的相关属性说明如表 13-4 所示。

参见如下的代码：

```
<s:iterator value="days">
    <p>星期几: <s:property/></p>
</s:iterator>
```

表 13-4 &lt;s:iterator&gt;标签的相关属性说明

属性名称	是否必须	默认值	数据类型	说 明
status	否	false	Boolean	如果被指定了, 则同名的变量 (类型为 IteratorStatus) 将放入值堆栈, 表示当前的迭代状态
value	否		String	被迭代的对象
var	否		String	放入值堆栈中的名称

根据上述这段代码，将会在值堆栈中查找名为 `days` 的属性，也就是调用 `getDays()` 方法得到要迭代的变量值（`days` 集合），然后再用 `<s:property>` 逐个输出 `days` 集合中的元素。再来看如下的代码：

```
<s:iterator status="stat" value="{1,2,3,4,5}" >
  <br>
  索引号: <s:property value="#stat.index" />&nbsp;
  是否奇数项: <s:property value="#stat.odd" />&nbsp;
  当前值为: <s:property/>&nbsp;
</s:iterator>
```

输出结果如下:

```
索引号: 0 是否奇数项: true 当前值为: 1
索引号: 1 是否奇数项: false 当前值为: 2
索引号: 2 是否奇数项: true 当前值为: 3
索引号: 3 是否奇数项: false 当前值为: 4
索引号: 4 是否奇数项: true 当前值为: 5
```

可见通过 `status` 属性中设置的变量 `stat`, `#stat.index` 表示迭代的当前元素索引号, 下标从 0 开始, `#stat.odd` 是否是第奇数个元素。

### 3. <s:append>标签

<s:append>标签用来做<s:iterator>标签的辅助,将不同 iterator 中的内容合并在一个 iterator 中。使用<s:append>标签的形式如下:

<s:append id="合并后的集合变量名称">

### 要合并的集合变量

&lt;/s:append&gt;

设有如下的一个 Action 类。

```
public class AppendIteratorTagAction extends ActionSupport {
    private List myList1;
    private List myList2;
    private List myList3;
    public String execute() throws Exception {
        myList1 = new ArrayList();
        myList1.add("1");
        myList1.add("2");
        myList1.add("3");

        myList2 = new ArrayList();
        myList2.add("a");
        myList2.add("b");
    }
}
```



```
myList2.add("c");

myList3 = new ArrayList();
myList3.add("A");
myList3.add("B");
myList3.add("C");

return SUCCESS;
}

public List getMyList1() { return myList1; }
public List getMyList2() { return myList2; }
public List getMyList3() { return myList3; }
}
```

如果在此 Action 相应的 JSP 页面中有如下的代码：

```
<s:append id="myAppendIterator">
    <s:param value="%{myList1}" />
    <s:param value="%{myList2}" />
    <s:param value="%{myList3}" />
</s:append>
<s:iterator value="%{#myAppendIterator}">
    <s:property />
</s:iterator>
```

则会一次性逐一将值堆栈中的 myList1、myList2、myList3 这三个集合对象的元素值输出，结果为：

```
1 2 3 a b c A B C
```

4. <s:generator>标签

<s:generator>标签用于根据其 val 属性直接生成一个 iterator 对象。使用此标签的形式如下：

```
<s:generator val="解析成 iterator 对象的源" separator="分隔符" [convert="转换成
object 对象的方法"] [id="变量名称"] [count="最大转换个数"]>
```

迭代操作语句：

```
</s:generator>
```

<s:generator>标签的相关属性说明如表 13-5 所示。

表 13-5 <s:generator>标签的相关属性说明

属性名称	是否必须	默认值	数据类型	说 明
converter	否		org.apache.struts2.util. IteratorGeneratorConverter	指定转换生成的 iterator 对象中的字符串元素成 object 对象的方法
count	否		Integer	val 中的值换成 iterator 对象的最大元素个数 s
var	否		String	将生成的 iterator 对象存储在 page 范围内的变 量名称
val	是		String	解析成 iterator 对象的源
separator	是		String	拆分 val 字符串成 iterator 对象的分隔符

属性 `converter` 的值如果设成 `xxxx`，则相应地会调用 `getXxxx()` 方法。

以下的一段代码：

```
<s:generator val="%{'aaa,bbb,ccc,ddd,eee'}" separator=",">
  <s:iterator>
    <s:property /><br/>
  </s:iterator>
</s:generator>
```

相当于生成了一个字符串 `iterator` 对象，然后再输出每个元素的值。再看如下的代码段：

```
<s:generator val="%{'aaa,bbb,ccc,ddd,eee'}" converter="%{myConverter}">
  <s:iterator>
    <s:property /><br/>
  </s:iterator>
</s:generator>
```

生成了一个字符串 `iterator` 对象，并调用相应的 `Action` 的 `getMyConverter()` 方法来做转换，再逐个输出字符串 `iterator` 对象中的元素。假设 `Action` 的代码如下：

```
public class GeneratorTagAction extends ActionSupport {
    ....
    public Converter getMyConverter() {
        return new Converter() {
            public Object convert(String value) throws Exception {
                return "converter-"+value;
            }
        };
    }
    ...
}
```

则表示将字符串 `iterator` 对象的每个元素的值在前面加入了“`converter-`”符号。

## 5. <s:merge>标签

`<s:merge>` 标签与 `<s:append>` 标签的功能有些类似，同样是将不同 `iterator` 中的内容合并在一个 `iterator` 中，但是两者合并的方式不同，故通过 `<s:iterator>` 标签显示合并后的 `iterator` 元素，结果会有所不同。

比如有 3 个集合，每个集合者均有 3 个元素，则使用 `<s:append>` 标签的合并顺序为：

第 1 个集合的第 1 个元素→第 1 个集合的第 2 个元素→第 1 个集合的第 3 个元素→第 2 个集合的第 1 个元素→第 2 个集合的第 2 个元素→第 2 个集合的第 3 个元素→第 3 个集合的第 1 个元素→第 3 个集合的第 2 个元素→第 3 个集合的第 3 个元素

使用 `<s:merge>` 标签的合并顺序为：

第 1 个集合的第 1 个元素→第 2 个集合的第 1 个元素→第 3 个集合的第 1 个元素→第 1 个集合的第 2 个元素→第 2 个集合的第 2 个元素→第 3 个集合的第 2 个元素→第 1 个集合的第 3 个元素→第 2 个集合的第 3 个元素→第 3 个集合的第 3 个元素

使用 `<s:merge>` 标签的形式如下：

```
<s:merge id="合并后的集合变量名称">
```

要合并的集合变量：



```
</s:merge>
```

来看个例子。如果仍然运用<s:append>标签中的 AppendIteratorTagAction 这个 Action 类,假定在 JSP 页面中的代码如下:

```
<s:merge id="myAppendIterator">
    <s:param value="%{myList1}" />
    <s:param value="%{myList2}" />
    <s:param value="%{myList3}" />
</s:merge>
<s:iterator value="%{#myAppendIterator}">
    <s:property />
</s:iterator>
```

则页面的输出结果如下:

```
1 a A 2 b B 3 c C
```

## 6. <s:subset>标签

<s:subset>标签用于获得可迭代集合的一个子集,以便后续处理。使用<s:subset>标签的形式如下:

```
<s:subset [count="取出的元素个数"]
    [decider="用于判断特定的元素是否包含在子集中" ]
    [id="结果子集的名称"] [source="源集合的名称"]
    [start="从 source 属性指定的集合的哪个位置开始截取"]>
```

迭代操作语句:

```
</s:subset>
```

<s:subset>标签的相关属性说明如表 13-6 所示。

表 13-6 <s:subset>标签的相关属性说明

属性名称	是否必须	默认值	数据类型	说 明
count	否		Integer	从 source 属性指定的集合中取出的元素个数
decider	否		org.apache.struts2.util._ SubsetIteratorFilter.Decider	用于判断特定的元素是否包含在子集中
var	否		String	将生成的 iterator 对象存储在 page 范围内的变量名称
source	否		String	源集合的名称
start	否		Integer	从 source 属性指定的集合的哪个位置开始截取

## 13.5.3 数据标签

### 1. <s:action>标签

通过<s:action>标签可以在 JSP 页面中直接调用 action。使用<s:action>标签的形式如下:

```
<s:action name="action 的名称" [executeResult="action 的 result 是否需要被执行"]
    [id="名称"] [namespace="action 所在的命名空间"]
    [ignoreContextParams="request 中的参数是否需要传入该 action"]/>
```

<s:action>标签的相关属性说明如表 13-7 所示。

表 13-7 <s:action>标签的相关属性说明

属性名称	是否必须	默认值	数据类型	说 明
name	是		String	指出要调用的 action 的名称
executeResult	否	false	Boolean	action 的 result 是否需要被执行
var	否		String	当前 action 在值堆栈中的名称
namespace	否		String	action 所在的命名空间
ignoreContextParams	否	false	Boolean	request 中的参数是否需要传入该 action

name 属性指出的 action 名称需要在 struts.xml 文件中做出配置, executeResult 属性也是根据 struts.xml 中的配置来知道决定执行的 result 是哪个页面。通过<s:action>标签也可以执行 Action 类中的某个特定的方法。参见如下实例。

#### 【实例 13-4】<s:action>标签使用示例

在本示例中将演示如何在 JSP 页面中直接调用 Action, 如何调用 Action 的特定方法。先编写如下所示的 Action 类。

##### ActionTagAction.java

```
package action;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class ActionTagAction extends ActionSupport {
    public String execute() throws Exception {
        return "done";
    }
    public String doDefault() throws Exception {
        ServletActionContext.getRequest().setAttribute("stringByAction",
            "这是执行 Action 的 doDefault() 方法时的输出!");
        return "done";
    }
}
```

这个 Action 类中声明了两个方法, execute()是默认情况下执行的方法; doDefault()是特定编写的方法。接下来在 struts.xml 文件中做出配置, 在<package>标签体中加入如下的配置:

```
<action name="actionTagAction1" class="action.ActionTagAction">
    <result name="done">success.jsp</result>
</action>
```

success.jsp 是当执行 Action 时, 如果返回 done, 所执行的 JSP 页面, 代码如下:

##### success.jsp

```
<%@ page contentType="text/html; charset=utf-8" %>
<html>
<body>
    执行成功的结果页面内容。
</body>
</html>
```



程序代码比较简单，就是输出一段文字“执行成功的结果页面内容。”接下来看调用 Action 的页面代码。

#### saction.jsp

```
<%@ page contentType="text/html;charset=utf-8" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<body>
<s:action name="actionTagAction1" executeResult="true" />
<br>
<s:action name="actionTagAction1!default" executeResult="false" />
<s:property value="#attr.stringByAction" />
</body>
</html>
```

第一个<s:action>标签调用 Action，并将 Action 的 result 中指定的 JSP 页面的执行结果放入当前页面；第二个<s:action>标签调用了 doDefault() 方法，之后再显示 request 范围中的 stringByAction 属性值。程序运行结果如图 13-13 所示。

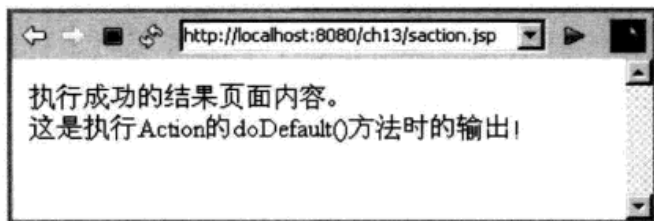


图 13-13 在 JSP 页面中调用 Action

## 2. <s:bean>标签

<s:bean>标签用于创建一个 JavaBean，并可利用本标签的标签体内的<s:param>标签来给 JavaBean 属性赋值，如果是 id 属性中的值，则将 JavaBean 以 id 属性指定的名称放入值堆栈中。使用<s:bean>标签的形式如下：

```
<s:bean name="类名(全路径)" [id="JavaBean 的名称"]>
    [<s:param>标签语句]
</s:bean>
```

## 3. <s:param>标签

此标签用于在其他标签中设置参数的值，如<s:include>标签、<s:bean>标签等。使用<s:param>标签的形式如下：

```
<s:param [name="属性名称"] [value="属性值"] />
```

或

```
<s:param [name="属性名称"]>
    属性值
</s:param>
```

第 1 种形式参数值格式为 java.lang.Object；第 2 种形式参数值格式为 String。

#### 4. <s:date>标签

此标签用于格式化日期数据，使用的形式如下：

```
<s:date name="要格式化的日期数据" [format="格式"]/>
```

#### 5. <si18n>标签

<s:i18n>标签用于创建一个与资源文件的绑定并放入值堆栈中，绑定后即可通过 text 标签来获取资源文件中的消息，使用的形式如下：

```
<s:i18n name="要绑定的资源文件的名称"/>
```

#### 6. <s:include>标签

<s:include>标签可用于包含其他的 JSP 页面或 Servlet，并可传入参数值，使用的形式如下：

```
<s:include value="JSP 页面或 Servlet 的名称"/>
```

或

```
<s:include value="JSP 页面或 Servlet 的名称">
    [<s:param>语句]
</s:include>
```

#### 7. <s:property>标签

此标签用于输出值堆栈中某个元素的值，如果没有明确指出，则默认情况下输出栈顶元素的值，使用形式如下：

```
<s:property [default="如果值为 null 时输出的默认值"]
    [escape="是否转换 HTML 特殊符号"] [value="要显示的元素"]/>
```

#### 8. <s:push>标签

<s:push>标签用于将值放入值堆栈中，以便于使用，使用形式如下：

```
<s:push value="放入值堆栈中的值"/>
```

或

```
<s:push value="放入值堆栈中的值">
    使用值的语句
</s:push>
```

#### 9. <s:set>标签

<s:set>标签用于设置一个特定范围内的值，使用形式如下：

```
<s:set [var="在值堆栈中的名称"] [scope="变量作用域"]
    [value="将会赋给变量的值"]/>
```

变量作用域可以是 application、session、request、page 或 action，默认值为 action。action 表示在 request 和 action 上下文范围内有效。

#### 10. <s:text>标签

<s:text>标签用于输出资源绑定文件中的消息，使用形式如下：



```
<s:text name="消息名称" [id="在值堆栈中的名称"] />
```

或

```
<s:text name="消息名称" [var="在值堆栈中的名称"] >
    消息内容
</s:text>
```

消息内容必须放在一个和当前 action 同名的资源文件中（扩展名为.properties），如果没有找到相应的消息则会采用标签体中的消息内容，如果标签体中也没有消息内容则会将 name 中的消息名称作为消息输出。

## 11. <s:url>标签

<s:url>标签用于生成一个 URL，使用的形式如下：

```
<s:url [action="用来生成 URL 的 action"] [anchor=" URL 的锚点"]
[encode="是否对参数进行解码"] [includeContext="是否包含上下文环境"]
[id="在值堆栈中的名称"] [method="调用的 Action 方法"]
[namespace="要使用的命名空间"] [value="目标地址"]>
    <s:param>语句
</s:url>
```

<s:url>标签的相关属性说明如表 13-8 所示。

表 13-8 <s:url>标签的相关属性说明

属性名称	是否必须	默认值	数据类型	说明
action	否		Object/String	指出用来生成 URL 的 action
anchor	否		String	URL 的锚点
encode	否	true	Boolean	是否对参数进行解码
includeContext	否	true	Boolean	实际的上下文环境是否应该包含在 URL 中
id	否		String	在值堆栈中的名称
method	否		Object/String	调用的 Action 方法
namespace	否		String	要使用的命名空间
value	否		String	不是使用 action 时的目标地址
includeParams	否	get	Object/String	值为'none'、'get' 或'all'

includeParams 属性值如果为 get 或 all，则优先采用<s:param>参数中的值。来看如下的代码：

```
<!--第 1 个例子-->
<s:url value="editGadget.action?id=2">
    <s:param name="id" value="'s'" />
</s:url>
<br>
<!--第 2 个例子-->
<s:url action="editGadget">
    <s:param name="id" value="'345'" />
</s:url>
<br>
<!--第 3 个例子-->
<s:url includeParams="get">
    <s:param name="id" value="%{'22'}" />
```

```
<s:param name="name" value="%{'希赛'}" />
</s:url>
```

生成的 HTML 代码如下所示：

```
editGadget.action?id=s
/chl3/editGadget.action?id=345
/chl3/saction.jsp?id=22&name=%CF%A3%C8%FC
```

对比即可发现：第 1 个例子有两个 id 参数，一个在 value 属性中，另一个是在<s:param>标签中，默认情况下 includeParams 属性的值为 get，故优先采用<s:param>标签中的参数值；第 2 个例子生成的是 action 的 URL，因此会自动在 action 名称后加上.action；第 3 个例子中有两个参数，其中有中文参数，故生成的 URL 参数中有%。


13.5.4 表单标签

表单标签有一些公共属性，常用的有：onchange（改变表单元素时的动作）、onclick（单击表单元素时的动作）、ondblclick（双击表单元素时的动作）、onfocus（获得焦点时的动作）、onselect（选中时的动作）等，在后述内容介绍时就不再一一列举了。

1. <s:checkbox>标签

<s:checkbox>标签用于生成 HTML 表单的复选框 checkbox，使用的形式如下：

```
<s:checkbox [id="checkbox 的 id 属性值"] [lable=" checkbox 的提示文字"]
        [lableSeparator="分隔符"] [name=" checkbox 的名称"]
        [fieldValue=" checkbox 的值"] [value="重新设置输入元素的值"]/>
```



**提示** 以上仅列出了常用的属性。

<s:checkbox>标签的常用属性说明如表 13-9 所示。

表 13-9 <s:checkbox>标签的常用属性说明

属性名称	是否必须	默认值	数据类型	说 明
id	否		String	生成 checkbox 的 id 属性值
lable	否		String	checkbox 的提示文字
labelSeparator	否	:	String	位于 checkbox 与提示文字之间的分隔符
labelposition	否		String	定义标签在 Form 中的位置，从左上计算
name	否		String	checkbox 的名称
fieldValue	否	true	String	checkbox 的值
value	否		String	重新设置输入元素的值

来看如下的代码。

```
<s:checkbox label="checkbox test" name="checkboxField1" value="%{true}"
fieldValue="true"/>
```

生成的 HTML 代码如下所示：

```
<tr>
```



```

<td valign="top" align="right">
    </td>
    <td valign="top" align="left">
<input type="checkbox" name="checkboxField1" value="true" checked="checked"
    id="checkboxField1"/>
<input type="hidden" name="__checkbox_checkboxField1" value="true" />
<label for="checkboxField1" class="checkboxLabel">checkbox test</label> </td>
</tr>

```

可见, 标签会自动生成相关的表格代码, 以及附属的隐藏输入域代码。复选框的提示文本位于右边, 当 value 属性值设为 true 时, 生成的 checkbox 会被选中, 即 checked="checked"。生成的代码显示如图 13-14 所示。

☒ checkbox test

图 13-14 生成的复选框

## 2. <s:checkboxlist>标签

<s:checkboxlist>标签用于生成一组复选框, 这些复选框名称相同, 但 id 不同。使用 <s:checkboxlist>标签的形式如下:

```

<s:checkbox list="用于设定 checkbox 组的列表" [id="checkbox 组的 id 属性值"]
[lable=" checkbox 组的提示文字"] [lableSeparator="分隔符"]
[labelposition="示文字位于 checkbox 组的方位"] [name=" checkbox 组的名称"]
[value="重新设置输入元素的值"]/>

```



**提示** 以上仅列出了常用的属性。

<s:checkboxlist>标签的常用属性说明如表 13-10 所示。

表 13-10 <s:checkboxlist>标签的常用属性说明

属性名称	是否必须	默认值	数据类型	说 明
id	否		String	生成 checkbox 组的 id 属性值
lable	否		String	Checkbox 组的提示文字
labelSeparator	否	:	String	位于 checkbox 组与提示文字之间的分隔符
labelposition	否		String	定义标签在 Form 中的位置, 从左上计算
list	是		String	指定用于设定 checkbox 组的列表
name	否		String	checkbox 组的名称
value	否		String	重新设置输入元素的值

来看如下的一段代码:

```

<jsp:useBean id="testList" class="java.util.ArrayList" scope="request"/>
<%
testList.add("a");
testList.add("b");
testList.add("c");

```

```
<%>
<s:set var="testList" value="#request.testList"/>
<s:checkboxlist name="check" list="testList"/>
```

这段代码先是生成一个 List 对象实例，范围为 request；在设置了值以后再利用<s:set>标签将 List 对象放入值堆栈。这段代码生成的 HTML 运行结果如图 13-15 所示。



图 13-15 <s:checkboxlist>标签的应用结果

3. <s:combobox>标签

此标签用于生成组合框，生成的组合框由一个文本输入框和一个下拉框组成，用户可以直接在文本输入框中输入值，也可以通过选择下拉框的选项来设置文本输入框的值。使用<s:combobox>标签的形式如下：

```
<s:combobox list="下拉框的列表" [id="文本框的 id 属性值"]
  [lable="组合框的提示文字"] [lableSeparator="分隔符"]
  [labelposition="示文字位于组合框的方位"] [name="文本框的名称"]
  [headerKey="下拉框第一个元素<option>的 value 属性值"]
  [headerValue="下拉框第一个元素<option>与</option>中间的文本"]
  [emptyOption ="是否生成空的<option>"] [value="重新设置输入元素的值"]/>
```

 **提示** 以上仅列出了常用的属性。

<s:combobox>标签的常用属性的说明如表 13-11 所示。

表 13-11 <s:combobox>标签的常用属性

属性名称	是否必须	默认值	数据类型	说明
headerKey	否		String	生成下拉框的第一个元素<option>的 value 属性值
headerValue	否		String	生成下拉框的第一个元素<option>与</option>中间的文本
id	否		String	生成文本框的 id 属性值
emptyOption	否	否	Boolean	是否生成空的<option>
lable	否		String	组合框的提示文字
lableSeparator	否	:	String	位于组合框与提示文字之间的分隔符
labelposition	否		String	定义标签在 Form 中的位置，从左上计算
list	是		String	指定用于设定下拉框的列表
listKey	否		String	用于设定下拉框<option>值的列表
listValue	否		String	用于设定下拉框<option>与</option>中间文本的列表
name	否		String	文本框的名称
value	否		String	重新设置输入元素的值

参见如下的代码：

```
<s:combobox id="testILove" label="我最喜欢的水果">
```



```
name="myFavouriteFruit" list="{ '苹果', '香蕉', '葡萄', '梨子' }"
headerKey="-1" headerValue="===请选择==="
emptyOption="false" value="香蕉" />
```

这段代码生成的 HTML 运行结果如图 13-16 所示:

图 13-16 组合框的应用结果

在选择了下拉框中的选项后,相应地就会自动修改文本输入框中的值;也可以直接修改文本输入框中的值。如何做到自动更新文本框中的数据呢?这是因为<s:combobox>标签自动生成了一些 JavaScript 代码,以上的代码生成的 HTML 代码如下所示:

```
<script type="text/javascript">
    function autoPopulate_testILove(targetElement) {
        if (targetElement.options[targetElement.selectedIndex].value == '-1') {
            return;
        }
        targetElement.form.elements['myFavouriteFruit'].value=
        targetElement.options[targetElement.selectedIndex].value;
    }
</script>
<input type="text" name="myFavouriteFruit" value="香蕉" id="testILove"/><br />
<select onChange="autoPopulate_testILove(this);">
    <option value="-1">===请选择===</option>
    <option value="苹果">苹果</option>
    <option value="香蕉" selected="selected">香蕉</option>
    <option value="葡萄">葡萄</option>
    <option value="梨子">梨子</option>
</select>
```



**提示** 以上代码需要在外再加上<form>这一 HTML 标签才能有效地进行事件响应。

#### 4. <s:doubleselect>标签

<s:doubleselect>标签用于生成两个联动的下拉框,其中第二个下拉框内容的变化依赖于第一个下拉框。<s:doubleselect>标签常用属性的说明如表 13-12 所示。

表 13-12 <s:doubleselect>标签常用属性的说明

属性名称	是否必须	默认值	数据类型	说明
doubleEmptyOption	否	否	Boolean	第二个下拉框是否生成一个空的<option>
doubleHeaderKey	否		String	第二个下拉框的第一个元素<option>的 value 属性值

续表

属性名称	是否必须	默认值	数据类型	说 明
doubleHeaderValue	否		String	第二个下拉框的第一个元素<option>与</option>中间的文本
doubleId	否		String	第二个下拉框的 id 属性值
doubleList	是		String	用于设定第二个下拉框的列表
doubleListKey	否		String	用于设定第二个下拉框<option>值的列表
doubleListValue	否		String	用于设定第二个下拉框<option>与</option>中间文本的列表
doubleName	是		String	第二个下拉框的名称
formName	否		String	当前标签所在表单的名称
emptyOption	否	否	Boolean	第一个下拉框是否生成一个空的<option>
headerKey	否		String	第一个下拉框的第一个元素<option>的 value 属性值
headerValue	否		String	第一个下拉框的第一个元素<option>与</option>中间的文本
id	否		String	第一个下拉框的 id 属性值
list	是		String	用于设定第一个下拉框的列表
listKey	否		String	用于设定第一个下拉框<option>值的列表
listValue	否		String	用于设定第一个下拉框<option>与</option>中间文本的列表
name	否		String	第 1 个下拉框的名称

来看下面的一段代码：

```
<form name="form1">
<s:doubleselect label="使用 doubleselect" name="menu" list="{ '湖南','广东' }"
formName="form1" doubleName="dishes"
doubleList="top=='湖南'?{'长沙','株洲'}:{'广州','佛山'}"/>
</form>
```

以上代码生成的 HTML 代码运行效果如图 13-17 所示。



图 13-17 联动下拉框

生成的 HTML 代码如下：

```
<form name="form1">
<select name="menu" id="menu"
onchange="menuRedirect(this.options.selectedIndex)">
<option value="湖南">湖南</option>
<option value="广东">广东</option>
</select>
<br />
<select name="dishes" id="dishes">
```



```

</select>
<script type="text/javascript">
    var menuGroup = new Array(2 + 0);
    for (i = 0; i < (2 + 0); i++)
        menuGroup[i] = new Array();
    menuGroup[0][0] = new Option("长沙", "长沙");
    menuGroup[0][1] = new Option("株洲", "株洲");
    menuGroup[1][0] = new Option("广州", "广州");
    menuGroup[1][1] = new Option("佛山", "佛山");
    var menuTemp = document.form1.dishes;
    menuRedirect(0);
    function menuRedirect(x) {
        var selected = false;
        for (m = menuTemp.options.length - 1; m >= 0; m--) {
            menuTemp.options[m] = null;
        }

        for (i = 0; i < menuGroup[x].length; i++) {
            menuTemp.options[i] = new Option(menuGroup[x][i].text,
                menuGroup[x][i].value);
        }
        if ((menuTemp.options.length > 0) && (! selected)) {
            menuTemp.options[0].selected = true;
        }
    }
}
</script>
</form>

```

## 5. <s:file>标签

<s:file>标签用于生成一个文件输入框。使用<s:file>标签的形式如下：

```
<s:file name="文件输入框的名称" accept="可接收文件的 MIME 类型"/>
```

## 6. <s:form>标签

<s:form>标签用于生成 HTML 中的<form>标签，使用形式如下：

```

<s:form [action="目标地址"] [enctype="表单的 enctype 属性值"]
    name="表单的名称" method="传输参数的形式"
    [onsubmit="提交表单时的动作"]/>

```

## 7. <s:hidden>标签

此标签用于生成隐藏起来的控件，使用形式如下：

```
<s:hidden name="隐藏域的名称" value="值"/>
```

## 8. <s:optiontransferselect>标签

此标签用于创建一个选项转移列表组件，它有两个<select ...>标签以及其间的用于将选项在两个<select ...>之间相互移动的按钮。表单提交时会自动选中全部选项。

<s:optiontransferselect>标签常用属性的说明如表 13-13 所示。

表 13-13 &lt;s:optiontransferselect&gt; 标签常用属性的说明

属性名称	是否必须	默认值	数据类型	说 明
doubleEmptyOption	否		Boolean	第二个 select 是否包含空的<option>选项
doubleHeaderKey	否		String	第二个 select 的第一个<option>元素的值
doubleHeaderValue	否		String	第二个 select 的第一个<option>与</option>之间的文本
doubleList	是		String	第二个 select 中的元素
list	是		String	第一个 select 中的元素
emptyOption	否		Boolean	第一个 select 是否包含空的<option>选项
headerKey	否		String	第一个 select 的第一个<option>元素的值
headerValue	否		String	第一个 select 的第一个<option>与</option>之间的文本

来看下面的代码：

```
<form name="form1">
<s:optiontransferselect
    name="leftSide"
    list="{ 'RFID 技术', 'GPS/GIS', '条形码' }"
    doubleName="rightSide"
    doubleList="{ '计算机应用', '网络管理', '数据库操作' }"
/>
</form>
```

生的 HTML 代码显示的结果如图 13-18 所示。

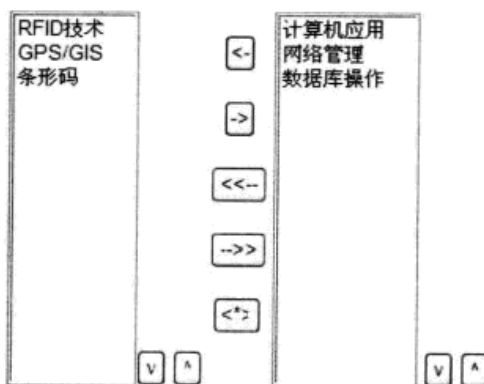


图 13-18 使用&lt;s:optiontransferselect&gt; 标签

自动生成的 HTML 代码如下。

```
<select name="leftSide" size="15" id="leftSide" multiple="multiple">
  <option value="RFID 技术">RFID 技术</option>
  <option value="GPS/GIS">GPS/GIS</option>
  <option value="条形码">条形码</option>
</select><input type="button"
  onclick="moveOptionDown(document.getElementById('leftSide'), 'key', '');"
  value="v"/>
<input type="button"
  onclick="moveOptionUp(document.getElementById('leftSide'), 'key', '');"
  value="^"/>
</td>
<td valign="middle" align="center">
```



```

        <input type="button" value="&lt;-";
        " onclick="moveSelectedOptions(document.getElementById('rightSide'),
        document.getElementById('leftSide'), false, ''); " /><br /><br />
        <input type="button" value="&->";
        onclick="moveSelectedOptions(document.getElementById('leftSide'),
        document.getElementById('rightSide'), false, ''); " /><br /><br />
        <input type="button" value="&lt;&lt;--";
        onclick="moveAllOptions(document.getElementById('rightSide'),
        document.getElementById('leftSide'), false, ''); " /><br /><br />
        <input type="button" value="--&gt;&gt;";
        onclick="moveAllOptions(document.getElementById('leftSide'),
        document.getElementById('rightSide'), false, ''); " /><br /><br />
        <input type="button" value="&lt;*&gt;";
        onclick="selectAllOptions(document.getElementById('leftSide'));
        selectAllOptions(document.getElementById('rightSide')); " /><br /><br />
</td>
<td>
<select name="rightSide" size="15" multiple="multiple" id="rightSide"
    <option value="计算机应用">计算机应用</option>
    <option value="网络管理">网络管理</option>
    <option value="数据库操作">数据库操作</option>
</select>
<input type="button"
    onclick="moveOptionDown(document.getElementById('rightSide'), 'key', ''); "
    value="v"/>
<input type="button"
    onclick="moveOptionUp(document.getElementById('rightSide'), 'key', ''); "
    value="^"/>

```

## 9. <s:optgroup>标签

`<s:optgroup>`标签用于生成`<select>`下拉框的`<option>`选项组，因此`<s:optgroup>`标签需要嵌套在`<s:select>`标签中使用。使用`<s:optgroup>`标签的形式如下：

```
<s:optgroup [list="生成选项组的集合"] [listKey="选项的值"]
    [listValue="选项的提示文本"]/>
```

来看如下的代码:

```
<s:select name="mySelection" list="%{#{'beijing': '北京', 'tianjing': '天津'}}">
  <s:optgroup label="湖南" list="%{#{'changsha': '长沙', 'zhuzhou': '株洲'}}" />
  <s:optgroup label="广东" list="%{#{'guangzhou': '广州', 'foshan': '佛山'}}" />
</s:select>
```

生成的 HTML 代码显示效果如图 13-19 所示。

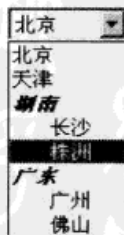


图 13-19 使用&lt;s:optgroup&gt;标签

这段代码生成的 HTML 代码如下：

```
<select name="mySelection" id="mySelection">
  <option value="beijing">北京</option>
  <option value="tianjing">天津</option>
  <optgroup label="湖南">
    <option value="changsha">长沙</option>
    <option value="zhuzhou">株洲</option>
  </optgroup>
  <optgroup label="广东">
    <option value="guangzhou">广州</option>
    <option value="foshan">佛山</option>
  </optgroup>
</select>
```

## 10. <s:password>标签

此标签用于生成密码输入框，使用的形式如下所示。

```
<s:password [lable="提示文本"] [name="输入框的名称"] [value="初始值"] />
```

读者可参看如下的示例语句：

```
<s:password name="mypass"/>
```

生成了一个名为 mypass 的密码输入框。

## 11. <s:radio>标签

此标签用于生成单选按钮，使用的形式如下：

```
<s:radio list="单选按钮的值及文本说明" [lable="提示文本"]
  [name="单选按钮的名称"] [value="初始值"] />
```

list 属性指出了单选按钮的值及文本说明，如果 list 属性设为 Map 对象，则 Map 对象的 key 会被设为单选按钮的值，value 会被设为单选按钮的文本说明。来看如下的代码：

```
<s:radio list="%{#{'1':'乒乓球','2':'篮球'}}" name="mylove1"/><br>
<s:radio list="%{#{'1':'乒乓球'}}" name="mylove2"/>
```

第 1 个语句生成了两个单选按钮，选择时只能选中其中的一个；第 2 个语句生成了一个单选按钮，如图 13-20 所示。

☐ 乒乓球 ☐ 篮球  
☐ 乒乓球

图 13-20 使用<s:radio>标签

以上两句程序代码生的 HTML 代码如下：

```
<input type="radio" name="mylove1" id="mylove11" value="1"/>
<label for="mylove11">乒乓球</label>
<input type="radio" name="mylove1" id="mylove12" value="2"/>
<label for="mylove12">篮球</label>
<br>
<input type="radio" name="mylove2" id="mylove21" value="1"/>
<label for="mylove21">乒乓球</label>
```



## 12. <s:reset>标签

<s:reset>标签生成重置按钮，使用的形式如下：

```
<s:reset [type="类型"] [label="提示文本"] [name="名称"] [value="初始值"] />
```

type 属性的值可以为 input 或 button，默认为 input，button 需要 IE6.0 以上版本才能支持。

## 13. <s:submit>标签

<s:submit>标签生成提交按钮，使用的形式如下：

```
<s:submit [type="类型"] [label="提示文本"] [name="名称"] [value="初始值"] />
```

type 属性的值可以为 input 或 button，默认为 input，button 需要 IE6.0 以上版本才能支持。

## 14. <s:select>标签

<s:select>标签用于生成下拉框，使用的形式如下：

```
<s:select list="下拉框的列表" [name="文本框的名称"]
    [label="组合框的提示文字"] [labelSeparator="分隔符"]
    [labelposition="示文字位于下拉框的方位"]
    [headerKey="下拉框元素<option>的 value 属性值"]
    [headerValue="下拉框元素<option>与</option>中间的文本"]
    [emptyOption="是否生成空的<option>"] [value="重新设置输入元素的值"] />
```

## 15. <s:textarea>标签

此标签用于生成文本域，使用的形式如下：

```
<s:textarea [label="提示文本"] [name="文本域的名称"] [value="初始值"]
    [cols="列数"] [rows="行数"] />
```

## 16. <s:textfield>标签

此标签用来生成文本输入框，使用的形式如下：

```
<s:textfield [label="提示文本"] [name="输入框的名称"] [value="初始值"] />
```

## 17. <s:updownselect>标签

此方法用于生成一个带有按钮的选择框组件，当表单提交时，位于上下两个被选择的元素之间（含被选择的元素）会被提交。比如如下的代码：

```
<s:updownselect
    list="%{#{'hunan':'湖南', 'guangdong':'广东', 'beijing':'北京'}}"
    name="province" headerKey="-1" headerValue="====请选择===="
    emptyOption="false" />
```

生成的 HTML 代码如下所示：

```
<tr><td>
<select name="province" size="5" id="province" multiple="multiple">
    <option value="-1">====请选择====</option>
    <option value="hunan">湖南</option>
    <option value="guangdong">广东</option>
    <option value="beijing">北京</option>
```

```

</select></td></tr>
<tr><td>
    &nbsp;<input type="button" value="^"
    onclick="moveOptionUp(document.getElementById('province'), 'key', '-1');" />
    &nbsp;&nbsp;<input type="button" value="v"
    onclick="moveOptionDown(document.getElementById('province'), 'key', '-1');" />
    &nbsp;&nbsp;<input type="button" value="*"
    onclick="selectAllOptionsExceptSome(document.getElementById('province'),
    'key', '-1');" />&nbsp;  
</td></tr>

```

显示效果如图 13-21 所示。

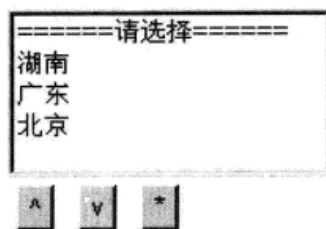


图 13-21 使用<s:updownselect>标签

### 13.5.5 非表单用户界面标签

非表单用户界面标签中需要掌握<s:actionerror>标签和<s:actionmessage>标签，这两个标签分别用于在 JSP 页面中显示来自 Action 中的错误和消息（有则显示，无则不显示），使用的方法如下：

```

<s:actionerror/>
<s:actionmessage/>

```

## 13.6 数据校验

Struts 2 提供了许多进行数据校验的方法，根据校验发生的场所可以分为服务端校验和客户端校验，根据校验框架的配置可以分为 Java 注释校验（不提倡使用）和 XML 配置文件校验。

客户端校验是指在 HTML 画面上自动生成 JavaScript 校验代码，在用户提交到服务器之前在客户端浏览器中进行校验。当然也可以是在 JSP 页面中自行编写 JavaScript 代码进行数据校验。服务端校验是指在数据提交到服务器上之后，在 Action 处理之前，对客户端提交的数据进行校验。

Java 注释校验是指使用 Java Annotation 语法，在 Java 源代码上标记需要校验的内容和校验的方式。XML 配置文件校验是指使用 XML 配置文件配置需要校验的内容和校验的方式。

本节接下来将详细讲解如何做数据校验，请读者注意体会校验属于哪一种校验方法，以及是如何进行校验的。



### 13.6.1 服务端和客户端数据校验

下面将详细讲解一个数据校验的应用实例，分别使用了服务端数据校验和客户端数据校验两种方法。

#### 【实例 13-5】用 XML 配置文件做简单的服务端和客户端数据校验

本例中 JSP 页面显示的是一个询问用户喜欢的颜色的表单，用户输入姓名、年龄、喜欢的颜色后提交表单，提交表单后即进行数据校验，如果数据校验出现错误，则返回输入页面。因此需要完成以下几项工作：

- (1) 编写输入数据的 JSP 页面。
- (2) 修改 struts.xml 文件，将本例的 action 配置加入。
- (3) 编写本例 Action 类的代码。
- (4) 编写 Action 对应的数据校验 XML 配置文件。

编写的用于输入数据的 JSP 页面源代码如下。

basicInput.jsp

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head><title>数据校验基础</title></head>
<body>
<b>问题：您最喜欢什么颜色？请回答。</b>
<s:form method="post" theme="xhtml"
    action="questionAction.action" validate="true" >
    <s:textfield name="name" label="姓名"/>
    <s:textfield name="age" label="年龄"/>
    <s:textfield name="answer" label="回答"/>
    <s:submit value="提交"/>
</s:form>
</body>
</html>
```

程序中声明了一个表单，数据提交的目标地址为 questionAction.action 这个 action，表单中有三个文本输入框、一个提交按钮。表单通过设置 theme 属性采用了 xhtml 主题，这样表单及表单中的输入标签会自动生成 HTML 表格标签。



**提示** <s:form>的 validate 属性值如果设置为 true，则 action 属性中给出的 Action 名称要加上“.action”，本例中为“questionAction.action”，否则会报出如下的错误：

```
Method public
java.util.List
org.apache.struts2.components.Form.getValidators(java.lang.String)
threw an exception when invoked on org.apache.struts2.components.
Form@117ee9e
The problematic instruction:
```

```

-----
==> list tag.getValidators("${tagName}") as validator [on line 46,
column 9 in template/xhtml/form-close-validate.ftl]
in include "${parameters.templateDir}/xhtml/form-close-validate.ftl"
[on line 25, column 1 in template/xhtml/form-close.ftl]
-----
... .. (后续报错内容)

```

basicInput.jsp 页面运行后的效果如图 13-22 所示。



图 13-22 basicInput 页面的运行效果

接下来，编辑 struts.xml 文件，在 “<struts>→<package>” 标签中加入如下的<action>配置。

```

<struts>
<package name="example" namespace="" extends="struts-default">
    ... ..
    <action name="questionAction" class="action.QuestionAction">
        <result name="success">success.jsp</result>
        <result name="input">basicInput.jsp</result>
    </action>
    ... ..
</package>
</struts>

```

从配置文件来看，questionAction 对应的 Action 类为 action 包中的 QuestionAction，如果数据校验通过，没有出现其他的错误，Action 的 execute() 方法（可以没有此方法）返回 success，则将跳转到 success.jsp；如果数据校验没有通过，将会跳转到 basicInput.jsp 页面并作报错处理。

QuestionAction 类的源代码如下。

#### QuestionAction.java

```

package action;
import com.opensymphony.xwork2.ActionSupport;
public class QuestionAction extends ActionSupport {
    String name;//姓名
    int age;//年龄
    String answer;//回答
    /**
    此处省去 name、age、answer 三个属性的 getXxxx() 方法和 setXxxx() 方法代码
    */
}

```



这个 Action 类中的代码相当简单，就是对应着 JSP 页面中的三个输入框就有三个属性，名称与 JSP 页面中的输入框名称相同，以一一对应，其中年龄为整型数据。Action 类中并没有重载 ActionSupport 的 execute() 方法。

接下来，继续编辑数据校验的 XML 配置文件。XML 配置文件需要存放在与 Action 类相同的目录中，比如本例中则在 action 包中新建一个 XML 文件，名称为 QuestionAction-validation.xml。给数据校验 XML 配置文件命名的规则是“Action 类名-validation.xml”。QuestionAction-validation.xml 文件的配置情况如下。

QuestionAction-validation.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
    "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <field name="name">
        <field-validator type="requiredstring">
            <message>必须输入姓名</message>
        </field-validator>
    </field>
    <field name="age">
        <field-validator type="int">
            <param name="min">18</param>
            <param name="max">30</param>
            <message>年龄只能是 18-30 岁之间</message>
        </field-validator>
    </field>
</validators>
```

从配置文件的情况来看，name 输入项数据验证的类型为 requiredstring，表示需要输入内容（不为空串），如果没有输入内容则报错信息为“必须输入姓名”；age 输入项数据校验的类型为 int，且最小值为 18，最大值为 30，如果不在 18~30 之间（含 18、30）则报错信息为“年龄只能是 18~30 岁之间”。

现在来做个操作，在 basicInput.jsp 界面中如果不输入内容，提交表单后结果如图 13-23 所示。

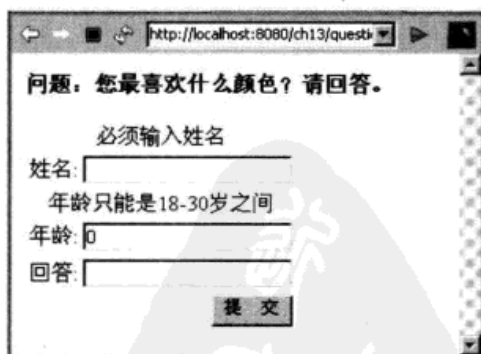


图 13-23 没有输入内容时返回 basicInput.jsp 页面的显示

以上做法是采取服务端数据校验的方法，如果要改成客户端校验，做法很简单，只需要修改一处地方。在 basicInput.jsp 页面中，<s:form>标签中加入一个属性 validate，值为 true，如下

所示。

```
<s:form method="post" theme="xhtml" action="questionAction" validate="true">
```

再次访问 `basicInput.jsp` 页面，可以发现系统自动生成了许多 JavaScript 代码。表单声明语句变为：

```
<form namespace="" id="questionAction" name="questionAction" onsubmit="return validateForm_questionAction();" action="/ch13/questionAction.action" method="post">
```

可见表单提交时通过 `validateForm_questionAction()` 函数来做数据校验，`validateForm_questionAction()` 函数的代码在页面的下方，如下所示。

```
<script type="text/javascript">
function validateForm_questionAction() {
    form = document.getElementById("questionAction");
    clearErrorMessages(form);
    clearErrorLabels(form);
    var errors = false;
    // field name: name
    // validator name: requiredstring
    if (form.elements['name']) {
        field = form.elements['name'];
        var error = "必须输入姓名";
        if (field.value != null && (field.value == "" ||
            field.value.replace(/^\s+|\s+$/g, "").length == 0)) {
            addError(field, error);
            errors = true;
        }
    }
    // field name: age
    // validator name: int
    if (form.elements['age']) {
        field = form.elements['age'];
        var error = "年龄只能是18-30岁之间";
        if (field.value != null) {
            if (parseInt(field.value) < 18 || parseInt(field.value) > 30) {
                addError(field, error);
                errors = true;
            }
        }
    }
    return !errors;
}
</script>
```

此外，页面中还有如下的语句：

```
<script type="text/javascript" src="/ch13/struts/xhtml/validation.js"></script>
```

以将 Struts 2 框架中的 `validation.js` 文件包含进来，其中的方法供本页面中生成的 JavaScript 代码使用。

### 13.6.2 字段校验

字段校验就是指对输入数据表单中的数据进行校验，由于表单中的一个（或多个）输入元



素对应着 Action 类中的一个（或多个）属性，故称为字段检验。Struts 2 内置了许多已有的常用检验规则供开发人员使用，实例 13-5 就是一个例子，比如对姓名的输入要求是一个字符串。字段校验在执行 Action 之前校验数据的合法性。下面来看一个综合使用 Struts 2 已有字段校验规则的实例。

### 【实例 13-6】使用 Struts 2 中的字段校验

首先编写一个 JSP 页面，用于输入数据，各个输入项对应可以应用 Struts 2 已有的各种字段校验规则，源代码如下。

fieldValidate.jsp

```
<%@ page contentType="text/html;charset=utf-8" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<title>字段校验</title>
</head>
<body>
```

所有的字段校验错误如下：

```
<s:fielderror />
<hr/>
```

字段内容没有输入错误的内容如下：

```
<s:fielderror>
<s:param value="%{'requiredStringValidatorField'}" />
</s:fielderror>
<hr/>
```

字段长度错误的内容如下：

```
<s:fielderror>
<s:param>stringLengthValidatorField</s:param>
</s:fielderror>
<hr/>
<s:form action="fieldValid" method="POST" theme="xhtml">
<s:textfield label="必须有的字段" name="requiredValidatorField" />
<s:textfield label="内容非空串数据校验" name="requiredStringValidatorField" />
<s:textfield label="整型数数据校验" name="integerValidatorField" />
<s:textfield label="日期型数据校验" name="dateValidatorField" />
<s:textfield label="Email 数据校验" name="emailValidatorField" />
<s:textfield label="URL 数据校验" name="urlValidatorField" />
<s:textfield label="字符串长度校验" name="stringLengthValidatorField" />
<s:textfield label="扩展名检验" name="regexValidatorField"/>
<s:textfield label="表达式数据校验" name="fieldExpressionValidatorField" />
<s:submit value="提 交" />
</s:form>
</body>
</html>
```

<s:fielderror/> 标签列出了所有字段校验产生的错误消息；如下的语句仅显示 'requiredStringValidatorField' 输入项数据校验产生的错误消息：

```
<s:fielderror>
    <s:param value="%{'requiredStringValidatorField'}" />
</s:fielderror>
```

输入数据的表单中有 9 个文本输入框，分别用于验证是否有内容非空串数据校验、整型数数据校验、日期型数据校验、Email 数据校验、URL 数据校验、扩展名检验、字符串长度校验以及表达式数据校验。JSP 页面的运行效果如图 13-24 所示。

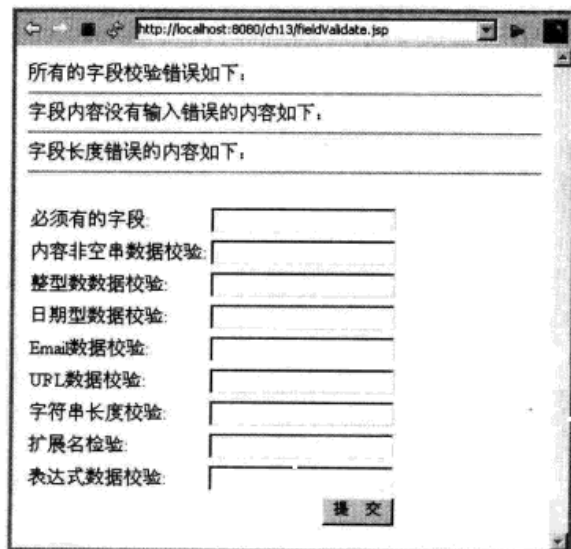


图 13-24 fieldValidate.jsp 页面的运行效果

编写 Action 类，其中的属性与输入数据的表单项一一对应，代码如下所示。

#### FieldValidAction.java

```
package action;
import java.util.Date;
import com.opensymphony.xwork2.ActionSupport;
public class FieldValidAction extends ActionSupport {
    private String requiredValidatorField = null;
    private String requiredStringValidatorField = null;
    private Integer integerValidatorField = null;
    private Date dateValidatorField = null;
    private String emailValidatorField = null;
    private String urlValidatorField = null;
    private String stringLengthValidatorField = null;
    private String regexValidatorField = null;
    private String fieldExpressionValidatorField = null;
    /**
```

此处省去以上属性的 getXxxx() 方法和 setXxxx() 方法代码

```
*/
```

```
}
```

从以上 Action 类的代码来看，属性与输入页面表单输入框的名称一一对应，而且具有对应的 setXxxx() 方法和 getXxxx() 方法。

接下来，编写校验数据的配置文件 FieldValidAction-validation.xml，这个文件与 Action 类位于同一目录下，配置内容如下。



## FieldValidAction-validation.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
    "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <field name="requiredValidatorField">
        <field-validator type="required">
            <message><![CDATA[需要有这个字段]]></message>
        </field-validator>
    </field>
    <field name="requiredStringValidatorField">
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message><![CDATA[需要输入内容而且必须是一个字符串]]></message>
        </field-validator>
    </field>
    <field name="integerValidatorField">
        <field-validator type="int">
            <param name="min">1</param>
            <param name="max">10</param>
            <message><![CDATA[如果有的话则必须是整数，且介于 1-10 之间
                (包括 1 和 10)]]></message>
        </field-validator>
    </field>
    <field name="dateValidatorField">
        <field-validator type="date">
            <param name="min">1990-01-01</param>
            <param name="max">2010-01-01</param>
            <message><![CDATA[日期型数据，如果有的话则必须位于 1990 年 1 月 1 日到 2010
                年 1 月 1 日之间]]></message>
        </field-validator>
    </field>
    <field name="emailValidatorField">
        <field-validator type="email">
            <message><![CDATA[如果有的话则必须是一个有效的 EMail]]></message>
        </field-validator>
    </field>
    <field name="urlValidatorField">
        <field-validator type="url">
            <message><![CDATA[如果有的话必须是一个有效的 URL]]></message>
        </field-validator>
    </field>
    <field name="stringLengthValidatorField">
        <field-validator type="stringlength">
            <param name="maxLength">4</param>
            <param name="minLength">2</param>
            <param name="trim">true</param>
            <message><![CDATA[如果有的话，至少有 2 个字符，最多 4 个字符]]></message>
        </field-validator>
    </field>
    <field name="regexValidatorField">
        <field-validator type="regex">
            <param name="expression">.*\.txt</param>

```



```

        <message><![CDATA[如果有的话, 扩展名必须为 txt ]]></message>
    </field-validator>
</field>
<field name="fieldExpressionValidatorField">
    <field-validator type="fieldexpression">
        <param name="expression">(fieldExpressionValidatorField ==
            requiredValidatorField)</param>
        <message><![CDATA[字段内容必须和
            requiredValidatorField 字段内容相同]]></message>
    </field-validator>
</field>
</validators>

```

以上代码对表单输入项的数据校验规则做了逐一设定。比如 `requiredStringValidatorField` 字段, 有如下的配置:

```

<field-validator type="requiredstring">
    <param name="trim">true</param>
    <message><![CDATA[需要输入内容而且必须是一个字符串]]></message>
</field-validator>

```

表示要输入的数据内容是一个字符串, `<param name="trim">true</param>` 表示对接收的字符串要调用的 `trim()` 方法去掉左右的空白字符, 结果不能为空串, 一旦出现错误则错误消息为“需要输入内容而且必须是一个字符串”。



**提示** XML 文件中的 `<![CDATA[... ...]]` 表示其间的内容是纯文本字符串。

再来看其中的一个配置例子:

```

<field-validator type="int">
    <param name="min">1</param>
    <param name="max">10</param>
    <message><![CDATA[如果有的话则必须是整数, 且介于 1-10 之间
        (包括 1 和 10) ]]></message>
</field-validator>

```

表示如果有输入数据的话, 则需要是整数, 且大小为 1 到 10 之间 (含 1 和 10)。这个数据校验的过程有两步, 第一步是 Struts 2 会对请求发来的接收数据自动做数据类型转换, 试图转换为整型数据, 一旦转换失败则会自动添加报错信息; 如果转换成功则进一步校验输入的数据是否位于 1 到 10 之间 (含 1 和 10), 如果不在此列, 则报错信息为“如果有的话则必须是整数, 且介于 1 到 10 之间 (包括 1 和 10)”。

下面随机输入一些数据并提交, 显示的结果如图 13-25 所示。

图 13-25 中在整型数输入框中输入了文本, 系统会自动做数据类型转换, 此时产生了异常, 错误提示文本为: `Invalid field value for field "integerValidatorField"`。也就是说, 先要自动进行数据类型转换, 转换成功后再根据配置文件中的配置做数据校验。

表达式校验在有些场合中相当有用, 比如在用户注册界面上要求两次输入密码, 且两次输入的密码相同, 则表达式与本例中的用法相同。



字段长度错误的代码如下:

必须有的字段:

内容非空串数据校验:

Invalid field value for field "integerValidatorField".

整型数据校验:

日期型数据校验:

如果有的话则必须是一个有效的EMail

Email数据校验:

如果有的话必须是一个有效的URL

URL数据校验:

字符串长度校验:

如果有的话, 扩展名必须为txt

扩展名校验:

字段内容必须和requiredValidatorField字段内容相同

表达式数据校验:

提交

图 13-25 本例数据校验的例子

针对数据类型的自动转换也可以在配置文件中做出专门的声明, 比如如下的配置 (放于数据校验配置文件中):

```
<!-- 专门的自动数据类型转换配置 -->
<validator type="conversion">
  <param name="fieldName">字段名称</param>
  <message>有错误发生时的报错信息</message>
</validator>
<!-- 自动数据类型转换配置放于字段数据校验配置之中 -->
<field name="myField">
  <field-validator type="conversion">
    <message>有错误发生时的报错信息</message>
  </field-validator>
</field>
```

前一种形式<validator>标签的 type 属性为 conversion, 表示是数据转换类型的数据校验, <param>标签的 name 属性指出标签体的内容为字段名称, <message>标签指出类型转换时有错误发生时的报错信息。

后一种形式<field>标签的 name 属性指出字段名称, <field-validator>标签的 type 属性值为 conversion, 表示是数据转换类型的数据校验, <message>标签指出类型转换时有错误发生时的报错信息。

这样做至少有两个作用: 一是重新设置自动数据类型转换时的报错信息, 因为默认情况下 Struts 2 的错误信息是英文, 需要换成中文; 二是可以用于保留原输入数据。比如要转换成整型数据, 如果原输入的是一个字符串, 则会自动转换出错, 跳转回输入页面后, 出现在输入框中的值却为 0, 如果要在输入框中保留原输入的字符串则需要参看如下的配置。

```
<validators>
  ...
  <field name="myIntegerField">
    <field-validator type="conversion">
      <param name="repopulateField">true</param>
```

```

        <message>数据类型自动转换出错</message>
    </field-validator>
</field>
...
</validators>

```

也就是说将参数 `repopulateField` 的值设为 `true`，则会保留原输入数据。

如果输入的数据类型要转换为浮点型数据，可以设置最小值、最大值，参见如下的配置：

```

<validators>
    <!-- 专门的配置 -->
    <validator type="double">
        <param name="fieldName">percentage</param>
        <param name="minInclusive">20.1</param>
        <param name="maxInclusive">50.1</param>
        <message> percentage 字段值必须>=${minInclusive},
            且<=${maxInclusive}</message>
    </validator>
    <!-- 配置放于字段数据校验配置之中 -->
    <field name="percentage">
        <field-validator type="double">
            <param name="minExclusive">0.123</param>
            <param name="maxExclusive">99.98</param>
            <message> percentage 字段值必须> ${minExclusive},
                且小于${maxExclusive}</message>
        </field-validator>
    </field>
</validators>

```

第一种形式采用的是闭区间，即包括最大值和最小值，消息中采用了参数化设置，比如 `${minInclusive}` 就代表值 20.1；第二种形式采用的是开区间，即不包括最大值和最小值。

Email 格式的自动数据校验（Email 是否有效）相当于使用了如下的正则表达式：

```

\\b(^[_A-Za-z0-9-](\\.[_A-Za-z0-9-])*(\\.[A-Za-z0-9-])+(\\.[\\.]com|\\.net|\\.org|\\.info|\\.edu|\\.mil|\\.gov|\\.biz|\\.ws|\\.us|\\.tv|\\.cc|\\.aero|\\.arpa|\\.coop|\\.int|\\.jobs|\\.museum|\\.name|\\.pro|\\.travel|\\.nato|\\.\\{2,3\\}|\\.\\{2,3\\}\\.\\{2,3\\})$)\\b

```

### 13.6.3 复杂的数据类型转换

Struts 2 内建支持一些数据类型的自动转换，开发人员没有必要再重新编写程序，直接用就可以了，其中包括：`String`、`boolean/Boolean`、`char/Character`、`int/Integer`、`float/Float`、`long/Long`、`double/Double`、`date`（使用 HTTP 请求对应 Locale 的 SHORT 形式转换字符串和日期类型）、`arrays`（每一个字符串内容可以被转换为不同的对象）、`collections`（转换为 `Collection` 类型，默认为 `ArrayList` 类型，其中包含 `String` 类型）。对于 `Array` 类型和 `Collection` 类型，需要对其中的每一个元素进行单独的转换。

下面来看一个稍复杂一些的处理实例，如何实现字符串向对象的转换。

#### 【实例 13-7】将字符串转换为对象

工程应用中可能常有这种需求：在输入框中输入一系列的数据，用某个符号相隔，提交数



据后需要转换为一个对象，后续处理直接操作对象即可。

因为 HTTP 协议传输的都是字符串数据，用户不可能在输入框中输入对象，需要开发人员编程来实现。比如，现在有如下的需求，用户在输入框中连续输入一个人的姓名和年龄，姓名和年龄之间用“,”相隔，数据提交后应将这个字符串转换为 **Person** 对象的姓名和年龄属性。

实现这个应用需要做以下工作：

- (1) 编写输入数据的 JSP 页面 **personInput.jsp**。
- (2) 编写 Action 类的程序 **PersonAction.java**。
- (3) 修改 Web 应用的 **struts.xml** 文件，增加这个应用的 action 配置。
- (4) 编写 **Person** 类的程序。
- (5) 编写 **PersonConvert** 类（数据转换器）的程序。
- (6) 编写转换对应关系的配置文件 **PersonAction-conversion.properties**。

输入数据的 JSP 页面比较简单，只需要一个输入框，一个提交按钮即可，源代码如下。

personInput.jsp

```
<%@ page contentType="text/html;charset=utf-8" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<body>
<s:actionerror/>
<s:form action="personInput" method="post" theme="xhtml">
    <s:textfield label="请输入一个人的姓名和年龄" name="person"/>
    <s:submit value="提交"/>
</s:form>
</body>
</html>
```

这个 JSP 页面运行的效果如图 13-26 所示。

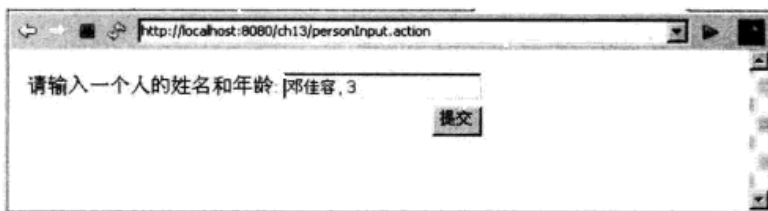


图 13-26 数据输入页面

Action 类 **PersonAction** 的源代码如下。

PersonAction.java

```
package action.person;
import com.opensymphony.xwork2.ActionSupport;
public class PersonAction extends ActionSupport{
    public Person person;
    public Person getPerson() {
        return person;
    }
    public void setPerson(Person person) {
        this.person = person;
    }
}
```

```

    }
    @Override
    public String execute() throws Exception {
        System.out.println(person);
        return "input";
    }
}

```

这个 Action 类由于与 Person 类处于同一个包中，故虽然用到了 Person 类也不需要使用 import 语句导入 Person 类就可以直接使用。类中有一个 person 属性，表示对应着输入界面中的 person 输入框；还有一个 execute() 方法，默认情况下这个方法在做完数据转换后会被自动执行，因此就使用 System.out.println() 方法来输出转换的结果，参数名 person 属性，也即调用 person 对象的 toString() 方法，execute() 方法返回 “input” 是为了跳转回 personInput.jsp 页面。

Person 类的代码如下所示。

#### Person.java

```

package action.person;
public class Person {
    public String name;
    public String age;
    /**
     * 此处省去 name、age 二个属性的 getXxxx() 方法和 setXxxx() 方法代码
     */
    public String toString() {
        StringBuffer sb = new StringBuffer("人(");
        sb.append(name).append(", ").append(age).append(")");
        return sb.toString();
    }
}

```

这个类有两个属性，分别表示姓名和年龄，还有一个 toString() 方法，这是重载了 Object 类的 toString() 方法，任何新建的类都继承自 Object 类。在用 System.out.print() 方法输出对象时，会自动调用对象的 toString() 方法。

person 输入框中的数据 and person 对象之间的转换，需要开发人员完成转换过程中的两项工作：一是输入数据字符串转换为 person 对象，二是 person 对象转换为数据字符串。至于如何获取字符串和返回字符串到输入框中，Struts 2 会自动完成这些工作。需要开发人员自行编写一个转换类来完成双向的转换工作，这个类的源代码如下。

#### PersonConvert.java

```

package action.person;
import java.util.Map;
import org.apache.struts2.util.StrutsTypeConverter;
public class PersonConvert extends StrutsTypeConverter {
    //从字符串转换为对象的方法
    public Object convertFromString(Map arg0, String[] arg1, Class arg2) {
        if (arg1.length > 0) {
            String personStr = arg1[0];
            String[] personStrArray = personStr.split(",");
            if (personStrArray.length == 2) {

```



```

        Person p = new Person();
        p.setName(personStrArray[0]);
        p.setAge(personStrArray[1]);
        return p;
    } else {
        return null;
    }
} else {
    return null;
}
}
//从对象转换为字符串的方法
public String convertToString(Map arg0, Object arg1) {
    if (arg1 instanceof Person) {
        return arg1.toString();
    } else {
        return "";
    }
}
}
}

```

这个类继承了 Struts 2 中的 `StrutsTypeConverter` 抽象类，需要实现 `convertFromString()` 和 `convertToString()` 这两个方法。下面还需要编写 `PersonAction-conversion.properties` 文件，这个文件必须与 Action 类在同一个目录中，即 `PersonAction`。转换配置文件的命名规则是“Action 类名-conversion.properties”。`PersonAction-conversion.properties` 文件的内容如下：

```

PersonAction-conversion.properties
person=action.person.PersonConverter

```

只有一行内容，等号前的 `person` 表示 Action 类中的 `person` 属性（或者说是输入框的名称），等号后的是转换器的类名（用全称）。

在输入如图 13-26 所示的数据后，按“提交”按钮，控制台有如图 13-27 所示的输出。

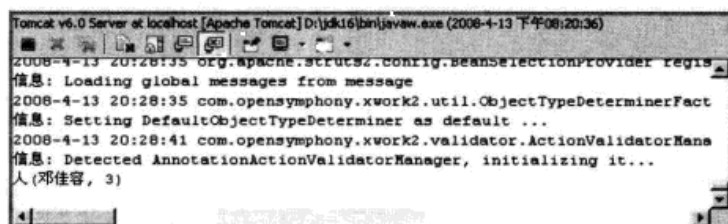


图 13-27 控制台的输出

可见，Action 类已经正确地转换了数据。然后程序会自动跳转回 `personInput.jsp` 页面，结果如图 13-28 所示。

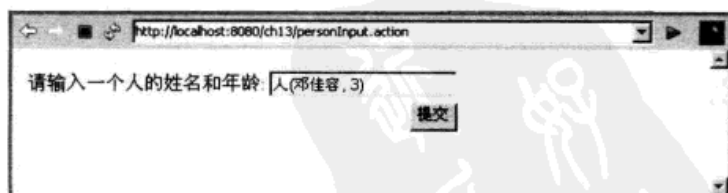


图 13-28 跳转回 `personInput.jsp` 页面的显示结果

回显的文字为“人（邓佳容，3）”，为什么和原字符串不同呢？仔细看看 `PersionConvert` 类的 `convertToString()` 方法就明白了，如果要转换为原字符串读者可修改这个方法里的代码。

如何将多个同名的文本输入框转换成 `List` 对象，这个并不难。比如有如下的一些同名的输入框：

```
<s:textfield label="请输入用户名" name="usernames"/>
<s:textfield label="请输入用户名" name="usernames"/>
<s:textfield label="请输入用户名" name="usernames"/>
<s:textfield label="请输入用户名" name="usernames"/>
<s:textfield label="请输入用户名" name="usernames"/>
```

对应地可在 `Action` 类中声明如下的属性：

```
public List usernames;
```

系统即会做出转换，而开发人员不必再行开发转换器。但是像实例 13-7 这样，如果有多个这样的同名输入框要输入，`Action` 对象是一个 `List` 对象集合，则仍然需要转换器；此外，还需要将转换配置文件修改为：

```
Element_persons=org.apache.struts2.showcase.conversion.Person
```

`Action` 类中的对应属性为：

```
public List persons;
```

JSP 页面中相应的输入代码修改为：

```
<s:iterator value="new int[3]" status="stat">
  <s:textfield label="%{'Person '+#stat.index+' Name'}"
    name="%{'persons['+#stat.index+'].name'}" />
  <s:textfield label="%{'Person '+#stat.index+' Age'}"
    name="%{'persons['+#stat.index+'].age'}" />
</s:iterator>
```

## 13.7 小结

Struts 2 是实现了 MVC 模式的框架。Struts 2 在 Struts 1 的基础上做了较大的改进，并融入了 WebWork 2。Struts 2 将一个 Web 系统的程序分为模型、视图、控制器三个部分，模型由 `JavaBean`、`EJB` 组件等完成具体业务的组件构成；视图由 JSP 文件、`POJO` 对象（可有可无）组成；控制器由 `Action` 来实现。

Struts 2 默认的表达式语言是 `OGNL`，`OGNL` 具有支持对象方法调用、支持静态方法调用等优点，能给 Struts 2 框架下的编程带来许多的便利。

Struts 2 标签库中的标签相当丰富，可以帮助开发人员简化 JSP 页面的程序编制工作。Struts 2 的标签分为通用标签和用户界面标签两大类。通用标签主要包括控制标签和数据标签；用户界面标签包括表单标签、非表单用户界面标签等。Struts 2 还提供了许多进行数据校验的方法，根据校验发生的场所可以分为服务端校验和客户端校验。



# 14

## 基于 Struts 2 实现 报到管理系统

本章中将和读者一道，采用 Struts 2 框架技术来改进第 12 章中实现的报到管理系统。通过学习本章的内容，将可以发现，应用 Struts 2 框架技术开发 Web 应用是相当简单的，而且具有很好的分层思想，在 JSP 页面中几乎看不到 Java 代码，下面就一起来领略吧。

### 14.1 系统设计思想

在第 12 章中用简单的 JSP 技术实现了报到管理系统，读者也能发现，这种技术存在如下的问题：

(1) 页面代码混乱。Java 代码与 HTML 夹杂在 JSP 页面中，当页面比较复杂、程序逻辑较多，程序的控制逻辑混杂在页面之中，代码看起来将感觉非常零乱。这样，一是需要开发人员有清晰的思路，对 Web 编程中哪些代码是运行在服务器端，哪些代码是运行的客户端理解透彻；二是给维护人员维护程序也带来相当的困难，对于复杂的页面，如果不是当初编写程序的程序员来维护，很难想象如何理解清楚程序代码中的逻辑。

(2) 安全性问题突出。一旦 JSP 页面被不怀好意的人看到源代码，或者网站被攻破，则 Java 代码完全暴露出来，带来了较大的安全隐患。

(3) 不适用于大型系统。主要原因是因为不利于开发人员之间的分工协作、代码共享，当系统规模增大时，工作量也急剧增加。

Struts 2 很好地解决了上述问题，它实现了 MVC 模式，并结合了 Struts 1 与 WebWork，博采众长，将表示层与业务逻辑区分开来，既保证了 JSP 页面的清爽（JSP 页面中大多采用标签来完成，很少直接使用 Java 代码），又使代码复用程序提高，如图 14-1 所示。

显示层 HTML 和 Struts 2 标签就能完成数据显示的任务，其中配合使用 OGNL 表达式。业

务逻辑层要完成的工作主要是数据校验和通过 Action 组件来做业务逻辑处理。比较简单的做法是直接将业务处理逻辑、数据处理逻辑代码都写在 Action 中，如果业务逻辑相当复杂，也可单独写在 JavaBean，再在 Action 中调用。

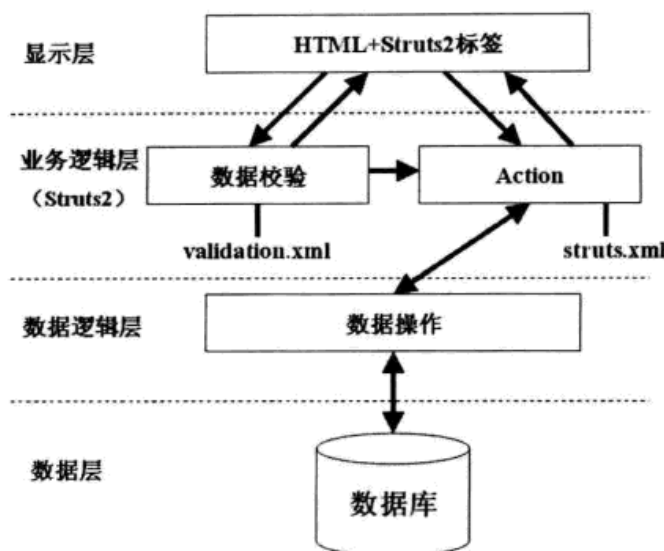


图 14-1 采用了 Struts 2 框架技术的系统架构

至于页面表单中的数据提交到哪个 Action，根据 Action 执行结果以跳转到哪个页面，这些控制流程都需要在 struts.xml 文件中做出配置，Struts 2 框架会根据这些配置来生成控制器来控制流程，而不需要开发人员自己去编写代码实现。

**提示** 数据库的设计与实现读者可参见第 12 章中的相关内容，由于数据结构都是一样的，本章中就不再赘述了。

## 14.2 系统开发框架搭建

### 14.2.1 在 Eclipse 中搭建 Web 应用的开发框架

在搭建 Web 应用的开发框架前，请读者先在机器上安装并配置好 JDK、Eclipse、Tomcat，准备好 Struts 2 开发包，在开发过程中需要使用到这些开发工具与软件包。

打开 Eclipse 后，选择“File”→“New”→“New Dynamic Web Project”菜单，会弹出“New Dynamic Web Project”对话框，如图 14-2 所示。

在“New Dynamic Web Project”对话框中输入工程名“ch14”，在“Project locaton”下设置工程文件放置的路径。在默认情况下会使用默认的设置，即为 Eclipse 工作区（workspace）的工作目录。不选中“Use default location”前的复选框即可使用下面的路径文本框和“Browse...”按钮来设置工程文件放置的目录，此时可以不采用系统的默认设置。

在“Target runtime”下设置 Web 应用的目标运行环境，此处为 Tomcat 7。单击“Finish”按钮完成 Web 应用框架的初步搭建。





图 14-2 新建一个动态 Web 工程

当前 Web 应用要使用到 Struts 2 框架, 因此请将 Struts 2 开发包的 lib 目录中的以下 jar 复制到当前 Web 应用的“WEB-INF\lib”目录中: commons-logging-1.1.1.jar、commons-logging-api-1.1.jar、freemarker-2.3.16.jar、ognl-3.0.1.jar、struts2-core-2.2.3.jar、xwork-2.2.3.jar、common-fileupload-1.2.2.jar、commons-io-2.0.1.jar、commons-lang-2.5.jar、javassist-3.11.0.GA.jar。

搭建完成的工程的 WEB-INF 文件夹情况如图 14-3 所示。

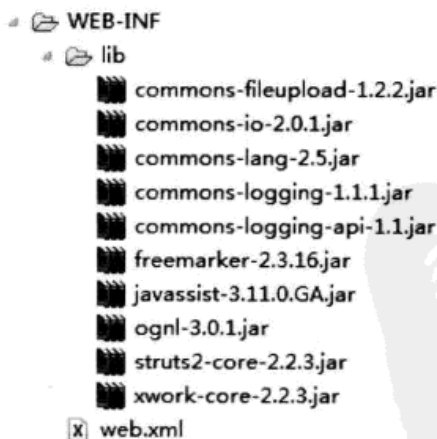


图 14-3 Web 应用框架搭建完成

生成 Web 工程后, Eclipse 已经将 JDK 和 Tomcat 自带的组件包都作为当前工程的组件包。build 目录中将存放编辑后的类的字节码文件。WebContent 是自动生成的存放 Web 应用文件的目录, Eclipse 已自动搭建好了 Web 应用的结构。

**提示** 复制完 jar 包以后, 注意要刷新后才能在 Eclipse 的树形菜单中看到更新的最新情况。刷新的方法是选中“ch14”节点, 按 F5 键, 或使用右键快捷菜单中的“刷新”菜单。

## 14.2.2 准备相关的配置文件与包

为简单起见, 本系统打算设置三个包: com.csai.action、com.csai.db、com.csai.POJO。com.csai.action 包中存放所有的 Action 类, com.csai.db 包存放与数据库操作相关的类, com.csai.POJO 包存放所有的 POJO 类。

新建 3 个包 com.csai.action、com.csai.db 和 com.csai.POJO, 比如新建 action 包的“New Java Package”对话框如图 14-4 所示。

在 src 节点上新建两个文件: struts.properties 和 struts.xml。struts.properties 的内容如下:

struts.properties

```
struts.locale=zh_CN
struts.i18n.encoding=GBK
```

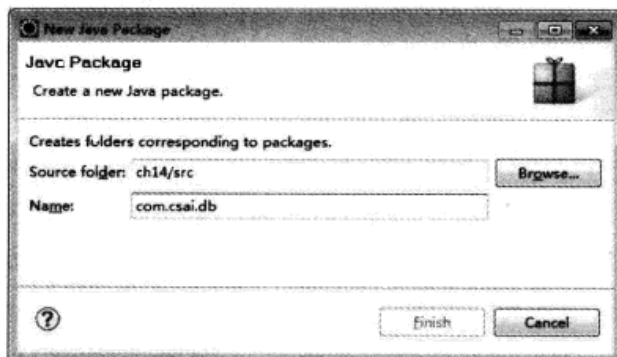


图 14-4 新建 com.csai.action 包

这样的话, 数据将会被解码为 GBK 编码, 即在所有程序中可以正确地处理中文字符, 不必再像 JSP 页面那样对每个接收到的参数都做编码转换处理了。

struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="ch14" namespace="" extends="struts-default">
        ... .. (有关 Action 的配置)
    </package>
</struts>
```



struts.xml 是 Struts 2 中的关键配置文件，用于控制程序处理的逻辑。<package>用于将当前 Web 应用的 Action 分包存放，可以给出命名空间的名称，这里采用默认的名称，extends="struts-default"表示继承自默认的配置，这种情况适合于大多数应用场合。

<action>标签用于配置各个 Action。在本章后述内容中对每个功能点的实现的详细说明中都会有相应的 Action 配置内容出现。

此外，还需要修改当前 Web 应用 WEB-INF 目录下的 web.xml 文件中的配置，内容如下。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>ch11</display-name>
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
    </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

在这个配置文件中声明了一个名称为“struts2”的过滤器，类为“org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter”，此过滤器与所有的 Web 请求相对应，这样 Web 应用中的请求响应中的处理就都能与 Struts 2 关联起来了。以上配置文件中“struts2”过滤器的配置在所有应用了 Struts 2 框架技术的 Web 应用中都是通用的。

## 14.3 系统各功能点的实现

下面一起来实现每个功能点。读者在阅读时可先参看前面几个功能是如何实现的，跟着一起来操作，在操作了前面的 2~3 个功能点后，后面功能点的实现直接阅读就能理解了，再操作起来就会比较熟练和快速了。



### 14.3.1 用户登录功能的实现

用户登录功能的实现要做以下的操作：开发一个登录页面 `login.jsp`，用户访问此页面时可输入用户名和密码，再按“提交”按钮，即在页面中做数据检查，如果用户名和密码正确则导向 `index.jsp` 页面，如果不正确则在 `login.jsp` 页面报错。

因为 login.jsp 页面是要建在 Web 应用的根目录中, 也即工程的 WebContent 目录中, 故先选中 Eclipse 左边树形菜单中的“WebContent”, 单击右键, 弹出快捷菜单, 选择“New”→“JSP File...”菜单, 弹出“New JSP File”对话框, 在“File name”后输入文件名“login.jsp”, 然后单击“Finish”按钮, 完成 login.jsp 的创建工作。

在 Eclipse 的代码编辑区中修改 login.jsp 的源代码，修改后的代码如下所示。

login.jsp

[illegible]





**提示** 编辑完 JSP 页面后此页面还不能运行成功，因为 Struts 2 中页面如果用到了 Struts 2 的表单标签，则框架会去检查是否有对应的 Action，因此请读者将后续步骤一并操作完毕后再调试程序。

对照第 12 章中的 login.jsp 页面即可发现，页面中不再有语句，代码相当简单，就是一些 Struts 2 的标签和 HTML 标签。要使用 Struts 2 的标签需要在页面的首部加入如下语句：

```
<%@ taglib prefix="s" uri="/struts-tags"%>
```

这样此后的代码中如何要使用 Struts 2 标签即可用 s 作为标签前缀。来看如下的语句：

```
<s:form method="post" action="Login" theme="simple">
```

action="Login"表明表单数据提交后，即与 Login 这个 Action 对应起来，至于 Action 对应的是哪个类，则要看 struts.xml 配置文件中的配置了。theme="simple"表明表单的主题使用的是 simple 主题，这个主题相对简单，不会自动生成 HTML 表格标签。



**提示** 为保持页面的美观，页面常常是由美工制作而成，因此采用 simple 主题是比较合适的，然而 Struts 2 默认采用的是 xhtml 主题，因此需要显示声明表单使用的是 simple 主题。

<s:fielderror/>语句可显示所有的字段错误，在与 Action 对应的校验数据的配置文件中和 Action 中均可产生字段错误。如下的语句：

```
<s:property value="errmsg"/>
```

这句话用于显示 Action 的 errmsg 属性值，这个属性打算用于设置登录验证过程中出现的各种错误消息，以利于将错误消息返回到 JSP 页面并显示。

在 struts.xml 文件中增加如下的配置：

```
<action name="Login" class="com.csai.action.LoginAction">
    <result>/index.jsp</result>
    <result name="input">/login.jsp</result>
</action>
```

name 属性的值为 Login，这与 JSP 页面表单的 action 属性值是一样的。class="com.csai.action.LoginAction" 表明 Action 对应着 com.csai.action.LoginAction 类。<result>/index.jsp</result>表明如果 Action 的 execute()方法返回 SUCCESS（此常量即为字符串“success”）即将页面导向至 index.jsp 页面，因为它相当于如下的语句：

```
<result name="success">/index.jsp</result>
```

如下的语句：

```
<result name="input">/login.jsp</result>
```

表明如果登录验证过程中出现错误或 Action 类的 execute()方法返回 INPUT（此常量即为字符串“input”），将页面导向至 login.jsp 页面。

在 Eclipse 左边的树形菜单中选中“com.csai.action”节点，单击右键，弹出快捷菜单，选择“New”→“Class”菜单，弹出“New Java Class”对话框。在“Name”后输入类名“LoginAction”，



单击“Finish”按钮，完成 LoginAction 类的创建。再在 Eclipse 的代码编辑器中修改 LoginAction 类的源代码，修改后代码如下。

## LoginAction.java

```
package com.csai.action;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import com.csai.db.DBConn;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String adminusername;//用户名
    public String adminuserpassword;//密码
    public String action;//操作类型
    public String errormsg;//错误消息

    @Override
    public String execute() {
        if("login".equals(action)){
            try{
                Connection conn=DBConn.createDBConn();
                String sql="select * from adminuser where adminusername=? "+
                    "and adminuserpassword=?";
                PreparedStatement state=conn.prepareStatement(sql);
                state.setString(1,adminusername);
                state.setString(2,adminuserpassword);
                ResultSet rs=state.executeQuery();
                if(rs.next()){//如果用户名和密码正确
                    ActionContext.getContext().getSession().put("adminusername",
                        adminusername);
                    ActionContext.getContext().getSession().put("adminusername",
                        adminusername);
                    ActionContext.getContext().getSession().put("adminuserrole",
                        rs.getString("adminuserrole"));
                    DBConn.closeConn(conn);
                    return SUCCESS;
                }else{
                    errormsg=new String("用户名或密码输入有误");
                }
                DBConn.closeConn(conn);
            }catch(Exception e){
                errormsg=new String("数据库连接有误");
            }
        }
        return INPUT;
    }
}
```

这个类的代码其实也简单。属性 adminusername 和属性 adminuserpassword 分别对应着登



录表单中的用户名和密码输入项。**action** 属性也是表单中提交来的数据项,表示要进行什么操作。**errmsg** 属性用于向 JSP 页面带回错误的消息。

在 **execute()**方法中就用于构造 SQL 语句,然后查询,如果有记录则表明用户名和密码是正确的,再通过 **ActionContext** 对象来向 **session** 中写入变量,记录下用户的用户名、密码和角色,以便于在其他程序中使用。

在进行数据库连接时使用到了 **com.csai.db.DBConn** 类,这个类的创建操作步骤是:在 Eclipse 左边的树形菜单中选中“**com.csai.db**”节点,单击右键,弹出快捷菜单,选择“**New**”→“**Class**”菜单,弹出“**New Java Class**”对话框。在“**Name**”后输入类名“**DBConn**”,单击“**Finish**”按钮,完成 **DBConn** 类的创建。再在 Eclipse 的代码编辑器中修改 **DBConn** 类的源代码,修改后代码如下。

#### DBConn.java

```
package com.csai.db;
import java.sql.Connection;
import java.sql.DriverManager;
public class DBConn {
    //得到数据库连接
    public static Connection createDBConn(){
        try{
            Connection conn=
                DriverManager.getConnection(
                    "jdbc:sqlserver://127.0.0.1:1433;
                    DatabaseName=RegisterSystem","sa","123");
            return conn;
        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }
    //关闭数据库连接
    public static void closeConn(Connection conn){
        try{
            conn.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

这个类中含有两个静态方法:一个用于得到一个数据库连接对象,一个用于关闭指定的数据库连接。专门设计一个 **DBConn** 类的好处是在其他程序代码中都可以使用这个类,而且如果数据库连接参数有变,则修改这个类中的代码即可。

在做用户数据校验时为简化开发,采用了配置文件来判断输入的数据是否为空,对应的 **Action** 数据检验配置文件如下。

#### LoginAction-validation.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
```

```

"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
  <field name="adminusername">
    <field-validator type="requiredstring">
      <param name="trim">true</param>
      <message><![CDATA[需要输入用户名]]></message>
    </field-validator>
  </field>
  <field name="adminuserpassword">
    <field-validator type="requiredstring">
      <param name="trim">true</param>
      <message><![CDATA[需要输入密码]]></message>
    </field-validator>
  </field>
</validators>

```

要求要输入用户名和密码，否则就产生字段型错误，并根据<message>标签中的文本来给出出错信息。<![CDATA[]]>表达其中的数据为纯文本数据。

### 14.3.2 专业基础数据管理功能的实现

专业基础数据管理功能要实现对专业表（Speciality）的增加、删除操作。specialityadmin.jsp文件的源代码如下。

specialityadmin.jsp

```

<%@ page contentType="text/html; charset=GBK" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<body>
<s:form method="post" action="Speciality" theme="simple">
<table border="1" cellpadding="0" cellspacing="0"
  style="border-collapse: collapse"
  bordercolor="#C0C0C0" width="600">
  <tr>
    <td width="100%" bgcolor="#C0C0C0">
      <font color="#0000FF">录入专业数据</font></td>
    </tr>
    <tr>
      <td width="100%">
        请输入专业名称:
        <s:textfield name="specialityname"/>
        <s:hidden name="action" value="add"/>
        <s:submit value="提交"/>
      </td>
    </tr>
  </table>
</s:form>
<table border="1" cellpadding="0" cellspacing="0"
  style="border-collapse: collapse"
  bordercolor="#C0C0C0" width="600">
  <tr>
    <td width="100%" bgcolor="#C0C0C0" align="center" colspan="3">
      <font color="#0000FF">已有专业数据</font></td>
    </tr>
  </table>

```



```

</tr>
<tr>
  <td width="20%" align="center">
    序号
  </td>
  <td width="60%" align="center">
    专业名称
  </td>
  <td width="20%" align="center">
    删除?
  </td>
</tr>
<s:iterator value="#request.specialityArray" status="status">
  <tr>
    <td width="20%" align="center">
      <s:property value="#status.count"/>
    </td>
    <td width="60%" align="center">
      <s:property value="specialityname"/>
    </td>
    <td width="20%" align="center">
      <a href="Speciality.action?action=del&specialityid=
<s:property value="specialityid"/>">
        删除? </a>
      </td>
    </tr>
  </s:iterator>
</table>
</body>
</html>

```

JSP 页面的上半部分用于录入数据，提交表单后新增一个专业，表单声明的语句如下：

```
<s:form method="post" action="Speciality" theme="simple">
```

表单将与 Speciality 这一 Action 对应起来。删除一个专业作的操 URL 地址为：

```
Speciality.action?action=del&specialityid=<s:property value="specialityid"/>
```

通过将 action 参数值设为 del 来告诉 Action 这是一个删除操作，并带去专业的 ID 号作为第二个参数。

来看 JSP 页面中的如下语句：

```
<s:iterator value="#request.specialityArray" status="status">
```

用于迭代输出 request 对象中的 specialityArray 这个 ArrayList 对象。specialityArray 在 Action 中被放入了 request 中，因此在 JSP 页面中可以通过 OGNL 表达式 #request.specialityArray 得到它。再来看看 Action 类的代码就清楚了。

#### SpecialityAction.java

```

package com.csai.action;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;

```



```

import java.util.ArrayList;
import java.util.Map;
import com.csai.POJO.Speciality;
import com.csai.db.DBConn;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class SpecialityAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String specialityname; //专业名称
    public int specialityid; //专业 ID 号
    public String action; //操作类型
    @Override
    public String execute() throws Exception {
        Connection conn=DBConn.createDBConn();
        //----如果是要增加一个专业---
        if("add".equals(action)){
            String sql="select * from Speciality where SpecialityName=?";
            PreparedStatement preSQLSelect=conn.prepareStatement(sql);
            preSQLSelect.setString(1,specialityname);
            ResultSet rs=preSQLSelect.executeQuery();
            if(!rs.next()){ //没有这个专业
                sql="insert into Speciality(SpecialityName) values(?)";
                PreparedStatement preSQLInsert=conn.prepareStatement(sql);
                preSQLInsert.setString(1,specialityname);
                preSQLInsert.executeUpdate();
            }
        }
        //----如果是删除一个专业----
        if("del".equals(action)){
            String sql="delete from Speciality where SpecialityId=?";
            PreparedStatement preSQLDel=conn.prepareStatement(sql);
            preSQLDel.setInt(1,specialityid);
            preSQLDel.executeUpdate();
        }
        //----查询出已有的专业数据----
        String sql="select * from Speciality";
        Statement state=conn.createStatement();
        ResultSet rs=state.executeQuery(sql);
        ArrayList<Speciality> specialityArray=new ArrayList<Speciality>();
        while(rs.next()){
            Speciality spec=new Speciality();
            spec.setSpecialityid(rs.getInt("specialityid"));
            spec.setSpecialityname(rs.getString("specialityname"));
            specialityArray.add(spec);
        }
        Map<String,ArrayList<Speciality>> request =
            (Map<String,ArrayList<Speciality>>)
                ActionContext.getContext().get("request");
        request.put("specialityArray", specialityArray);
        DBConn.closeConn(conn);
    }
}

```



```

        return SUCCESS;
    }
    /**
     * 此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}

```

并不是每个表单提交的数据非得与 Action 类的属性一一对应, 如果 Action 类中的属性在表单中没有对应的输入项, 则 Action 类中的该属性值为 null, 这样一个 Action 类就可以对应多个请求了, 比如本例中的增加专业和删除专业的操作都被封装在 SpecialityAction 中了。

在 Action 类中通过如下的语句得到了 request 对象:

```

Map<String, ArrayList<Speciality>> request =
    (Map<String, ArrayList<Speciality>>)
    ActionContext.getContext().get("request");

```

然而这里的 request 对象不同于 JSP 页面中的 request 对象, 它只是一个简单的 Map 对象, 但是对它的操作会影响到 request 对象中的参数值。以上的语句使用了 Java 中的泛型而将 request 对象中的参数类型设置为 ArrayList<Speciality>, 也就是说 Map 中的 key 对应的 value 数据类型为 ArrayList<Speciality>。ArrayList<Speciality> 表示 ArrayList 中的每个元素类型均为 Speciality, Speciality 是一个 POJO 对象, 属性和方法很简单, 就是专业 ID 号、专业名称及其对应的 setXxxx() 和 getXxxx() 方法。

以上语句得到的 request 对象只适用于向其中加入 ArrayList<Speciality> 参数, 如果产生一个通过的 request, 可加入各种数据类型的参数, 则不使用泛型即可, 语句如下:

```

Map request = (Map)ActionContext.getContext().get("request");

```

得到 request 对象后, 即可使用 put() 方法向其中加入参数, 这样在 JSP 页面中就可通过 OGNL 表达式 #request.refname 的形式得到这个参数。



**提示** POJO 类的代码相当简单, 此处就不再列出了。

本功能在 struts.xml 文件中对应的配置内容如下:

```

<action name="Speciality" class="com.csai.action.SpecialityAction">
    <result>/basicdata/specialityadmin.jsp</result>
</action>

```

### 14.3.3 录取学生名册基础数据管理功能的实现

录取学生名册基础数据管理功能的操作界面如图 12-25 所示。为集成地在一个 JSP 页面中录入、查询、删除操作, 操作界面也设计得相对复杂一些。来看 JSP 页面的代码。

matri.jsp

```

<%@ page contentType="text/html; charset=GBK" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<body>
<s:form method="post" action="Matri" theme="simple">

```

```

<table border="1" cellpadding="0" cellspacing="0"
        style="border-collapse:collapse"
        bordercolor="#C0C0C0" width="600">
    <tr>
        <td width="100%" bgcolor="#C0C0C0">
            <font color="#0000FF">录入录取学生名册</font></td>
        </tr>
    <tr>
        <td width="100%" align="left">
            请输入学生姓名:
            <s:textfield name="studentname"/>
            请选取录取专业:
            <s:select name="specialityid" listKey="specialityid"
                listValue="specialityname" list="#request.specialityArray"
                headerKey="" headerValue=="请选择==">
            </s:select><br>
            请输入录取通知书号: <s:textfield name="matrino"/>
            <s:hidden name="action" value="add"/>
            <s:submit value="提交"/>
        </td>
    </tr>
</table>
</s:form>
<s:form method="post" action="Matri" theme="simple">
<table border="1" cellpadding="0" cellspacing="0"
        style="border-collapse:collapse"
        bordercolor="#C0C0C0" width="600">
    <tr>
        <td width="100%" bgcolor="#C0C0C0" colspan="5">
            <font color="#0000FF">查询已录入的学生名册</font></td>
        </tr>
    <tr>
        <td colspan="5">
            请输入学生姓名:
            <s:textfield name="studentname"/>
            请选取录取专业:
            <s:select name="specialityid" listKey="specialityid"
                listValue="specialityname" list="#request.specialityArray"
                headerKey="" headerValue=="请选择==">
            </s:select>
            <s:hidden name="action" value="select"/>
            <s:submit value="提交"/>
        </td>
    </tr>
    <tr>
        <td align="center" colspan="5"><font color="#0000FF">
            <s:if test="#request.pagecount>1&&#request.currentpage>1">
                <a href="Matri.action?currentpage=1&action=<s:property value="action"/>
                    &specialityid=<s:property value="specialityid"/>
                    &studentname=<s:property value="studentname"/>">首页</a>&nbsp;
                <a href="Matri.action?currentpage=<s:property
                    value="#request.currentpage-1"/>
                    &action=<s:property value="action"/>

```



```

        &specialityid=<s:property value="specialityid"/>
        &studentname=<s:property value="studentname"/>">上一页</a>&nbsp;
    </s:if>
    <s:if test="#request.pagecount>1"
        &&#request.currentpage<#request.pagecount">
        <a href="Matri.action?currentpage=<s:property
            value="#request.currentpage+1"/>
            &action=<s:property value="action"/>
            &specialityid=<s:property value="specialityid"/>
            &studentname=<s:property value="studentname"/>">下一页</a>&nbsp;
        <a href="Matri.action?currentpage=<s:property
            value="#request.pagecount"/>
            &action=<s:property value="action"/>
            &specialityid=<s:property value="specialityid"/>
            &studentname=<s:property value="studentname"/>">尾页</a>&nbsp;
    </s:if>
    共<s:property value="#request.recount"/>条记录,
    共<s:property value="#request.pagecount"/>页,
    当前第<s:property value="#request.currentpage"/>页
</font></td>
</tr>
<tr bgcolor="#C0C0C0">
    <td align="center"><font color="#0000FF">序号</font></td>
    <td align="center"><font color="#0000FF">姓名</font></td>
    <td align="center"><font color="#0000FF">录取专业</font></td>
    <td align="center"><font color="#0000FF">录取通知书号</font></td>
    <td align="center"><font color="#0000FF">删除? </font></td>
</tr>
<s:iterator value="#request.stuArray" status="status">
    <tr>
        <td align="center"><s:property value="#status.count"/></td>
        <td align="center">
            <s:property value="StudentName"/>
        </td>
        <td align="center">
            <s:property value="@com.csai.db.StudentUtil@haveSplitSpec
                (SpecialityId)"/>
        </td>
        <td align="center">
            <s:property value="MatriNo"/>
        </td>
        <td align="center">
            <a href="Matri.action?action=del&studentid=<s:property
                value="StudentId"/>">删除</a>
        </td>
    </tr>
</s:iterator>
</table>
</s:form>
</body>
</html>

```

估计读者经过前两个功能中代码的分析也能看懂以上 JSP 文件中的大部分代码了, 下面分析一些关键的、难懂的语句。



```
<s:select name="specialityid" listKey="specialityid"
    listValue="specialityname" list="#request.specialityArray"
    headerKey="" headerValue=="请选择==">
</s:select>
```

以上用于显示表示专业的下拉框。list 属性指出专业列表对象，可以是 Map 或 Collection 对象，如果是 Collection 对象则应给出 listKey 和 listValue 属性，以让 Struts 2 知道使用 Collection 中对象的哪个属性作为下拉框 option 的 value 属性值，哪个属性作为下拉框 option 的提示文本。

这里在 Action 中将所有的专业数据取出，放入到一个 ArrayList 中，ArrayList 的元素类型为 Speciality 这个 POJO 类，故采用 Speciality 类的 specialityid 属性名作为<s:select>标签的 listKey 属性值，specialityname 属性名作为<s:select>标签的 listValue 属性值。

headerKey 属性指出显示在下拉框中第一个元素的值，headerValue 指出显示在下拉框中第一个元素的提示文本。



**提示** 读者可自行对照 Action 类的代码，再体会一下 OGNL 表达式和<s:select>标签的用法，以加深认识。

```
<s:property value="@com.csai.db.StudentUtil@haveSplitSpec(SpecialityId)"/>
```

这个语句调用了 com.csai.db.StudentUtil 类的静态方法 haveSplitSpec()，传入专业 ID 号（是迭代的 Speciality 对象的 SpecialityId 属性），从而得到表述专业名称的字符串。来看 StudentUtil 类的代码。

#### StudentUtil.java

```
package com.csai.db;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
public class StudentUtil {
    //返回是否已分专业表述的字符串
    public static String haveSplitSpec(int specialityid){
        String returnString;
        try{
            Connection conn=DBConn.createDBConn();
            if(specialityid==0)
                returnString="尚无专业";
            else{
                String sql="select * from speciality
                    where specialityid="+specialityid;
                Statement statetemp=conn.createStatement();
                ResultSet rstemp=statetemp.executeQuery(sql);
                if(rstemp.next())
                    returnString=rstemp.getString("specialityname");
                else
                    returnString="尚无专业";
            }
            DBConn.closeConn(conn);
            return returnString;
        }catch(Exception e){
```



```

        return null;
    }
}
//返回是否已分班表述的字符串
public static String haveSplitClass(int classid){
    String returnString;
    try{
        Connection conn=DBConn.createDBConn();
        if(classid==0)
            returnString="尚未分班";
        else{
            String sql="select * from classTa where classid="+classid;
            Statement statetemp=conn.createStatement();
            ResultSet rstemp=statetemp.executeQuery(sql);
            if(rstemp.next())
                returnString=rstemp.getString("classname");
            else
                returnString="尚未分班";
        }
        DBConn.closeConn(conn);
        return returnString;
    }catch(Exception e){
        return null;
    }
}
//返回是否已分配宿舍表述的字符串
public static String haveSplitBedchamber(int bedchamberid){
    String returnString;
    try{
        Connection conn=DBConn.createDBConn();
        if(bedchamberid==0)
            returnString="尚未分配宿舍";
        else{
            String sql="select * from bedchamber where bedchamberid="+
                bedchamberid;
            Statement statetemp=conn.createStatement();
            ResultSet rstemp=statetemp.executeQuery(sql);
            if(rstemp.next())
                returnString=rstemp.getString("bedchambername");
            else
                returnString="尚未分配宿舍";
        }
        DBConn.closeConn(conn);
        return returnString;
    }catch(Exception e){
        return null;
    }
}
}

```

这个类给出了三个静态的方法，`haveSplitSpec()`方法根据专业 ID 号得到专业的名称；`haveSplitClass()`方法根据班级 ID 号得到班级名称；`haveSplitBedchamber()`方法根据宿舍 ID 号得到宿舍名称。



接下来编写 struts.xml 文件中有关 Action 的配置内容。

```
<action name="Matri" class="com.csai.action.MatriAction">
<result>/basicdata/matri.jsp</result>
</action>
```

由以上代码可知对应的 Action 类为 com.csai.action.MatriAction, 下面再来看 Action 类的代码。

#### MatriAction.java

```
package com.csai.action;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Map;
import com.csai.POJO.Speciality;
import com.csai.POJO.Student;
import com.csai.db.DBConn;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class MatriAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String action;//操作类型
    public int specialityid;//专业 ID 号
    public String matrino;//录取通知书号
    public String studentname;//学生姓名
    public int currentpage=1;//当前页页码
    public long studentid;//学生 ID 号
    @Override
    public String execute() throws Exception {
        if(studentname!=null&&studentname.length()!=0)
            studentname=studentname.trim();
        if(action!=null&&action.length()!=0)
            action=action.trim();
        if(matrino!=null&&matrino.length()!=0)
            matrino=matrino.trim();
        Connection conn=DBConn.createDBConn();
        //----如果是增加一个学生---
        if("add".equals(action)){
            String sql="select * from Student where matrino=? and studentname=?"+
                "and specialityid=?";
            PreparedStatement preSQLSelect=conn.prepareStatement(sql);
            preSQLSelect.setString(1,matrino);
            preSQLSelect.setString(2,studentname);
            preSQLSelect.setInt(3,specialityid);
            ResultSet rs=preSQLSelect.executeQuery();
            if(!rs.next()){//没有这个专业
                sql="insert into Student (matrino,studentname,specialityid) "+
                    " values(?,?,?)";
                PreparedStatement preSQLInsert=conn.prepareStatement(sql);
                preSQLInsert.setString(1,matrino);
                preSQLInsert.setString(2,studentname);
                preSQLInsert.setInt(3,specialityid);
```



```

        preSQLInsert.executeUpdate();
    }
}
//----如果是要删除一个学生---
if("del".equals(action)){
    String sql="delete from Student where studentid=?";
    PreparedStatement preSQLUpdate=conn.prepareStatement(sql);
    preSQLUpdate.setLong(1,studentid);
    preSQLUpdate.executeUpdate();
}
//----如果是要查询数据----
ResultSet rsselect=null;
int pagesize=5;//每页记录条数
int pagecount=0;//总页数
int recount=0;//总记录条数
if("select".equals(action)){
    String sql=null;
    if((specialityid==0)){
        sql="select top "+pagesize+" studentname,"+
            "studentid,matrino,specialityid "+
            "from Student where 1=1 ";
        String sqlcount="select count(*) as recount from Student ";
        Statement stateCount=conn.createStatement();
        ResultSet rscount=stateCount.executeQuery(sqlcount);
        rscount.next();
        recount=rscount.getInt("recount");//得到总记录条数
        //----得到总页数----
        if(recount%pagesize==0)//能整除
            pagecount=recount/pagesize;
        else//不能整除
            pagecount=(int)(recount/pagesize)+1;
        //----生成得到当前页数据查询的附件条件----
        if(pagecount>1&&currentpage>1){
            String sqladd=" and studentid not in "+
                "(select top "+(currentpage-1)*pagesize+
                " studentid from Student "+
                " order by studentid desc) ";
            sql=sql+sqladd;
        }
        sql=sql+" order by studentid desc ";
        Statement state=conn.createStatement();
        rsselect=state.executeQuery(sql);
    }else{
        //----先生成查询的 where 子句----
        String sqlwhere=" where 1=1 ";
        if(specialityid!=0)//如果选择了专业
            sqlwhere=sqlwhere+" and specialityid=? and
                studentname like ? ";
        else//如果没有选择专业
            sqlwhere=sqlwhere+" and studentname like ? ";
        sql="select top "+pagesize+" studentname,studentid,matrino,
            specialityid "+"from Student "+sqlwhere;
    }
}

```



```

//----得到总记录条数----
String sqlcount="select count(*) as recount "+
    "from Student "+sqlwhere;
PreparedStatement preSQLCount=conn.prepareStatement(sqlcount);
if(specialityid!=0){//如果选择了专业
    preSQLCount.setInt(1,specialityid);
    preSQLCount.setString(2,"%"+studentname+"%");
}else
    preSQLCount.setString(1,"%"+studentname+"%");
ResultSet rscount=preSQLCount.executeQuery();
rscount.next();
recount=rscount.getInt("recount");//得到总记录条数
//----得到总页数----
if(recount%pagesize==0)//能整除
    pagecount=recount/pagesize;
else//不能整除
    pagecount=(int)(recount/pagesize)+1;
//----生成得到当前页数据查询的附件条件----
PreparedStatement preSQLSelect=null;
//总页数大于1且当前页码大于1
if(pagecount>1&&currentpage>1){
    String sqladd=" and studentid not in "+
        "(select top "+(currentpage-1)*pagesize+
        " studentid from Student "+sqlwhere+
        "order by studentid desc) ";
    sql=sql+sqladd;
    sql=sql+" order by studentid desc ";
    preSQLSelect=conn.prepareStatement(sql);
    if(specialityid!=0){//如果选择了专业
        preSQLSelect.setInt(1,specialityid);
        preSQLSelect.setString(2,"%"+studentname+"%");
        preSQLSelect.setInt(3,specialityid);
        preSQLSelect.setString(4,"%"+studentname+"%");
    }else{//如果没有选择专业
        preSQLSelect.setString(1,"%"+studentname+"%");
        preSQLSelect.setString(2,"%"+studentname+"%");
    }
}else{//当前为第1页或仅只有1页
    sql=sql+" order by studentid desc ";
    preSQLSelect=conn.prepareStatement(sql);
    if(specialityid!=0){//如果选择了专业
        preSQLSelect.setInt(1,specialityid);
        preSQLSelect.setString(2,"%"+studentname+"%");
    }else
        preSQLSelect.setString(1,"%"+studentname+"%");
}
rsselect=preSQLSelect.executeQuery();
}
}
ArrayList<Student> stuArray=new ArrayList<Student>();
while(rsselect!=null&&rsselect.next()){
    Student stu=new Student();
    stu.setStudentName(rsselect.getString("studentname"));
}

```



```

        stu.setSpecialityId(rsselect.getInt("specialityid"));
        stu.setMatriNo(rsselect.getString("matrino"));
        stu.setStudentId(rsselect.getLong("studentid"));
        stuArray.add(stu);
    }
    Map request = (Map)ActionContext.getContext().get("request");
    request.remove("stuArray");
    request.put("stuArray", stuArray);
    //----查询出专业数据----
    String sql="select * from Speciality";
    Statement state=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
    ResultSet rs=state.executeQuery(sql);
    ArrayList<Speciality> specialityArray=new ArrayList<Speciality>();
    while(rs.next()){
        Speciality spec=new Speciality();
        spec.setSpecialityid(rs.getInt("specialityid"));
        spec.setSpecialityname(rs.getString("specialityname"));
        specialityArray.add(spec);
    }
    request.remove("specialityArray");
    request.put("specialityArray", specialityArray);
    //----将数据放入 request----
    request.remove("pagesize");
    request.remove("pagecount");
    request.remove("currentpage");
    request.remove("recount");
    request.put("pagesize",pagesize);
    request.put("pagecount",pagecount);
    request.put("currentpage",currentpage);
    request.put("recount",recount);
    DBConn.closeConn(conn);
    return SUCCESS;
}
/**
 *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
 */
}

```

凡是在 JSP 页面中需要显示的动态数据都应当在 Action 中取出来,因此这里 Action 类代码还是比较长的,其实仔细读读也不难,无非还是像第 12 章中的那样,构造 SQL 语句,执行 SQL 语句,得到查询结果,将查询结果组成方便 JSP 页面显示的数据类型(比如将 ResultSet 转换、组合成 ArrayList)放入到 request 中。

在编写 Action 时,开发人员可以专心将业务逻辑写清楚,而不需要去考虑 Web 页面的格式问题。



**提示** 如何分页的原理就不再详细讲述了,请查阅第 12 章中对应的本功能实现的详解,在这里只不过是 JSP 页面中分页处理的代码迁移到 Action 中来罢了,然后将需要在 JSP 页面中显示的数据放入到 request 对象中。

### 14.3.4 其他基础数据管理功能的实现

宿舍基础数据管理、班级基础数据管理、用户管理功能的实现方法和专业基础数据管理功能的实现方法相同，也就不再重复叙述了，读者可在读懂“12.4.2 专业基础数据管理功能的实现”一节的基础上再直接查看宿舍基础数据管理、班级基础数据管理功能的实现源代码就会很容易了。

### 14.3.5 学生报到状况查询功能的实现

在学生报到查询功能中，在查询条件中输入学生的姓名，单击“提交”按钮，即会在下面的表格中查询出所要查询的学生的报到状况。报到的状况包括分班情况、交费情况、所在宿舍的情况。

regstatus.jsp

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<body>
<s:form method="post" action="RegStatus" theme="simple">
<table border="1" cellpadding="0" cellspacing="0" style="border-collapse: collapse"
bordercolor="#C0C0C0" width="700">
<tr>
<td width="100%" bgcolor="#C0C0C0">
<font color="#0000FF">录入要查询状态的条件</font></td>
</tr>
<tr>
<td width="100%">
请输入姓名:
<s:textfield name="studentname"/>
<s:hidden name="action" value="select"/>
<s:submit value="提交"/>
</td>
</tr>
</table>
</s:form>
<table border="1" cellpadding="0" cellspacing="0" style="border-collapse: collapse"
bordercolor="#C0C0C0" width="700">
<tr>
<td width="100%" bgcolor="#C0C0C0" align="center" colspan="8">
<font color="#0000FF">已有的学生报到数据</font></td>
</tr>
<tr>
<td width="5%" align="center">序号</td>
<td width="12%" align="center">姓名</td>
<td width="16%" align="center">录取通知书号</td>
<td width="12%" align="center">录取专业</td>
<td width="15%" align="center">所在班级</td>
<td width="10%" align="center">是否交费</td>
```



```

        <td width="15%" align="center">已交学费</td>
        <td width="15%" align="center">所在宿舍</td>
    </tr>
    <s:set name="stuArray" value="#request.stuArray" scope="action"/>
    <s:if test="#stuArray!=null">
        <s:iterator value="#stuArray" status="status">
            <tr>
                <td align="center"><s:property value="#status.count"/></td>
                <td align="center"><s:property value="StudentName"/></td>
                <td align="center"><s:property value="MatriNo"/></td>
                <td align="center">
                    <s:property value="@com.csai.db.StudentUtil@haveSplitSpec(SpecialityId)"/>
                </td>
                <td align="center">
                    <s:property value="@com.csai.db.StudentUtil@haveSplitClass(ClassId)"/>
                </td>
                <td align="center">
                    <s:if test="PayOK==0">
                        未交清
                    </s:if>
                    <s:else>
                        已交清
                    </s:else>
                </td>
                <td align="center"><s:property value="PayAmount"/></td>
                <td align="center">
                    <s:property value="@com.csai.db.StudentUtil@haveSplitBedchamber
                        (BedchamberId)"/>
                </td>
            </tr>
        </s:iterator>
    </s:if>
</table>
</body>
</html>

```

以上代码也并不复杂，读者在阅读以上代码时难度可能主要来自于阅读 **struts2** 标签和 **OGNL** 表达式。比如在 **OGNL** 表达式中什么时候用“#”符号，什么时候不需要，有时把握不准。一是要看 **struts2** 标签属性的数据类型，如果是 **Object**，则可以直接使用 **OGNL** 表达式，如果是 **String**，则 **OGNL** 表达式求值就要使用“%{}”符号，不论如何，如果要在 **struts2** 标签属性中对 **OGNL** 表达式求值，均可使用“%{}”。二是要看当前的值堆栈情况，如果是 **Action** 的属性，则直接用属性名，如果是迭代中的标签，要得到被迭代对象的属性也可直接使用属性名，综合来说就是如果不是当前对象的属性则应使用“#”符号。读者如果对以上表述还是觉得不好理解，那么您就只好在实践中去多应用多体会了。

本功能在 **struts.xml** 配置文件中的 **Action** 配置内容如下：

```

<action name="RegStatus" class="com.csai.action.RegStatusAction">
    <result>/basicdata/regstatus.jsp</result>
</action>

```

由此可见对应的 **Action** 类为 **RegStatusAction**，来看它的代码。



## RegStatusAction.java

```

package com.csai.action;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Map;
import com.csai.POJO.Student;
import com.csai.db.DBConn;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class RegStatusAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String studentname;//学生姓名
    public String action;//操作类型
    @Override
    public String execute() {
        if("select".equals(action)){
            //----查询出已有的数据----
            try{
                Connection conn=DBConn.createDBConn();
                String sql="select * from student ";
                if(studentname!=null&&studentname.length()!=0)
                    sql+="where studentname like '%'+studentname+'%'";
                Statement state=conn.createStatement();
                ResultSet rs=state.executeQuery(sql);
                ArrayList<Student> stuArray=new ArrayList<Student>();
                while(rs.next()){
                    Student stu=new Student();
                    stu.setBedchamberId(rs.getInt("bedchamberId"));
                    stu.setClassId(rs.getInt("classId"));
                    stu.setMatriNo(rs.getString("matriNo"));
                    stu.setPayAmount(rs.getFloat("payAmount"));
                    stu.setPayOK(rs.getInt("payOK"));
                    stu.setRegistDate(rs.getDate("registDate"));
                    stu.setSpecialityId(rs.getInt("specialityId"));
                    stu.setStudentId(rs.getLong("studentId"));
                    stu.setStudentName(rs.getString("studentName"));
                    stuArray.add(stu);
                }
                Map<String,ArrayList<Student>> request = (Map<String,
                    ArrayList<Student>>)ActionContext.getContext().get("request");
                request.put("stuArray", stuArray);
                DBConn.closeConn(conn);
            }catch(Exception e){
                e.printStackTrace();
            }
        }
        return SUCCESS;
    }
    /**
     * 此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}

```



此外没有再使用多表联合查询的 SQL 语句了，因为在 JSP 页面中使用了简单的 OGNL 表达式就可调用静态方法来得到专业名称、班级名称之类的表述字符串了。

### 14.3.6 报到分班功能的实现

报到分班功能中有一个难点，查询结果的记录条数是不确定的，那么表单中输入项的个数相应地也就不能确定，那么在 Struts 2 中又要如何来接收分班的数据并进行分班处理呢？在 Struts 2 框架中实现这些功能比在 JSP 中要方便得多了，因为数据类型的映射支持 Map、Collection 等，甚至还可以自定义。

classadmin.jsp

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<body>
<s:form method="post" action="ClassAdmin" theme="simple">
<table border="1" cellpadding="0" cellspacing="0"
    style="border-collapse: collapse"
    bordercolor="#C0C0C0" width="700">
<tr>
<td width="100%" bgcolor="#C0C0C0">
<font color="#0000FF">要查询的条件</font></td>
</tr>
<tr>
<td width="100%">
    请输入姓名:
    <s:textfield name="studentname"/>
    请输入录取通知书号:
    <s:textfield name="matrino"/>
    <s:hidden name="action" value="select"/>
    <s:submit value="提交"/>
</td>
</tr>
</table>
</s:form>
<s:form action="ClassAdmin" method="post" theme="simple">
<table border="1" cellpadding="0" cellspacing="0"
    style="border-collapse: collapse"
    bordercolor="#C0C0C0" width="700">
<tr>
<td width="100%" bgcolor="#C0C0C0" align="center" colspan="8">
<font color="#0000FF">查询到的学生数据</font></td>
</tr>
<tr>
<td width="5%" align="center">序号</td>
<td width="12%" align="center">姓名</td>
<td width="16%" align="center">录取通知书号</td>
<td width="12%" align="center">录取专业</td>
<td width="15%" align="center">所在班级</td>
<td colspan="3"></td>
<td colspan="2"></td>
</tr>
<s:iterator value="#request.stuArray" status="status">
<tr>
```

```

        <td width="5%" align="center"><s:property value="#status.count"/></td>
        <td width="12%" align="center"><s:property value="StudentName"/></td>
        <td width="16%" align="center"><s:property value="MatriNo"/></td>
        <td width="12%" align="center">
        <s:property value="@com.csai.db.StudentUtil@haveSplitSpec(SpecialityId)"/>
        </td>
        <td width="15%" align="center">
        <s:hidden name="stuParamArray[%{#status.index}].StudentId"
            value="%{StudentId}"/>
        <s:select name="stuParamArray[%{#status.index}].ClassId" listKey="classid"
            listValue="classname" list="#request.classArray"
            headerKey="0" headerValue=="请选择==" value="ClassId">
        </s:select>
        </td>
    </tr>
</s:iterator>
<tr>
    <td width="100%" bgcolor="#C0C0C0" align="center" colspan="8">
        <font color="#0000FF">
        <s:submit value="确定"/>
        </font></td>
</tr>
<s:hidden name="action" value="update"/>
</table>
</s:form>
</body>
</html>

```

参看如下的语句:

```
<s:iterator value="#request.stuArray" status="status">
```

`stuArray` 是在 `Action` 中得出的查询结果学生 `ArrayList`, 使用 `<s:iterator>` 来迭代 `ArrayList` 对象最合适不过了。一起来看其中比较难于理解的语句。

为了让 `Action` 得到要分班的人的 ID 号和分班后的班级 ID 号, 在迭代体中使用了如下的标签语句:

```

<s:hidden name="stuParamArray[%{#status.index}].StudentId" value="%{StudentId}"/>
<s:select name="stuParamArray[%{#status.index}].ClassId" listKey="classid"
    listValue="classname" list="#request.classArray"
    headerKey="0" headerValue=="请选择==" value="ClassId">
</s:select>

```

`<s:hidden>` 标签的 `name` 属性是字符串数据类型, 然而这里并不要求值, 只是需要将名称与 `Action` 对应起来。比如要将迭代中的一连串的学生 ID 号赋给 `Action` 中的一个 `ArrayList` 属性, 如何对应呢? 就是通过名称了。`stuParamArray` 是 `Action` 类中的一个 `ArrayList` 类型的对象, “`%{#status.index}`” 这个表达式的结果就是索引号。下拉框的命令方式也是如此。

接下来看在 `Action` 类中是如何做的, 代码如下。

#### ClassAdminAction.java

```

package com.csai.action;
import java.sql.Connection;

```



```

import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Map;
import com.csai.POJO.ClassTa;
import com.csai.POJO.Student;
import com.csai.db.DBConn;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class ClassAdminAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String studentname;
    public String action;
    public String matrino;
    public ArrayList<Student> stuParamArray;
    @Override
    public String execute() throws Exception {
        Connection conn=DBConn.createDBConn();
        //----构造查询的 SQL 语句----
        String sqlwhere=new String("");
        String sql=new String("");
        if("select".equals(action)){//如果是查询操作
            if(studentname!=null&&studentname.trim().length()!=0)
                sqlwhere="where studentname like '%" +studentname.trim()+"%' ";
            if(sqlwhere!=null&&sqlwhere.length()!=0){
                if(matrino!=null&&matrino.trim().length()!=0)
                    sqlwhere+=" and matrino like '%" +matrino.trim()+"%' ";
            }else{
                if(matrino!=null&&matrino.trim().length()!=0)
                    sqlwhere=" where matrino like '%" +matrino.trim()+"%' ";
            }
            sql="select * from student "+sqlwhere;
            Statement state=conn.createStatement();
            ResultSet rs=state.executeQuery(sql);
            ArrayList<Student> stuArray=new ArrayList<Student>();
            while(rs!=null&&rs.next()){
                Student stu=new Student();
                stu.setStudentName(rs.getString("studentname"));
                stu.setSpecialityId(rs.getInt("specialityid"));
                stu.setMatriNo(rs.getString("matrino"));
                stu.setStudentId(rs.getLong("studentid"));
                stu.setClassId(rs.getInt("classid"));
                stuArray.add(stu);
            }
            Map request = (Map)ActionContext.getContext().get("request");
            request.put("stuArray", stuArray);
            //----查询出已有的专业数据----
            sql="select * from ClassTa";
            Statement stclass=conn.createStatement();
            ResultSet rsclass=stclass.executeQuery(sql);
            ArrayList<ClassTa> classArray=new ArrayList<ClassTa>();
            while(rsclass.next()){
                ClassTa classta=new ClassTa();
                classta.setClassid(rsclass.getInt("classid"));

```



```

        classta.setClassname(rsclass.getString("classname"));
        classArray.add(classta);
    }
    request.put("classArray", classArray);
}
//-----设置分班情况-----
if(stuParamArray!=null&&"update".equals(action)){
    for(int i=0;i<stuParamArray.size();i++){
        if(stuParamArray.get(i).getClassId()!=0){
            String sqlstr="update student set classid="
                +stuParamArray.get(i).getClassId()+
                "where studentid="+stuParamArray.get(i).
                getStudentId();
            Statement state=conn.createStatement();
            state.executeUpdate(sqlstr);
        }
    }
    DBConn.closeConn(conn);
    return SUCCESS;
}
/**
 *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
 */
}

```

stuParamArray 属性的类型为 ArrayList<Student>, 使用了泛型, 这样 Struts 2 就清楚地知道是何种数据类型了。设置分班的情况也就是生成更新的 SQL 语句, 再执行这些语句。

分班情况查询功能则相对比较简单, 与学生报到情况查询功能实现方法类似, 只是查询条件换成了班级, 即按班级查询。此外, 收费登记情况查询、宿舍分配情况查询功能所调用的 JSP 页面与分班情况查询功能的相同, 均是按班级查询报到中的情况。读者可自行查看随书光盘中的源代码。

### 14.3.7 收费情况登记功能的实现

收费情况登记的业务流程与报到分班功能中的实现非常相似。在查询条件表格中输入要查询的学生姓名、录取通知书号, 按“提交”按钮, 即将查询的结果列在下面的表格中。针对学生报到的情况, 将会出现交费情况登录的录入框。

如果还没有分班, 则不能交纳学费, 如果已经分班, 则可录入交费金额, 设置学费是否交清单选按钮。下面是页面的源代码。

```

                                acceptmoney.jsp
<%@ page contentType="text/html;charset=GBK" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<body>
<s:form method="post" action="AcceptMoney" theme="simple">
<table border="1" cellpadding="0" cellspacing="0" style="border-collapse: collapse"
    bordercolor="#C0C0C0" width="700">

```



```

<tr>
  <td width="100%" bgcolor="#C0C0C0">
    <font color="#0000FF">要查询的条件</font></td>
</tr>
<tr>
  <td width="100%">
    请输入姓名:
    <s:textfield name="studentname"/>
    请输入录取通知书号:
    <s:textfield name="matrino"/>
    <s:hidden name="action" value="select"/>
    <s:submit value="提交"/>
  </td>
</tr>
</table>
</s:form>
<s:form method="post" action="AcceptMoney" theme="simple">
<table border="1" cellpadding="0" cellspacing="0"
  style="border-collapse:collapse"
  bordercolor="#C0C0C0" width="700">
  <tr>
    <td width="100%" bgcolor="#C0C0C0" align="center" colspan="7">
      <font color="#0000FF">查询到的学生数据</font></td>
    </tr>
    <tr>
      <td width="5%" align="center">序号</td>
      <td width="12%" align="center">姓名</td>
      <td width="16%" align="center">录取通知书号</td>
      <td width="12%" align="center">录取专业</td>
      <td width="15%" align="center">所在班级</td>
      <td width="15%" align="center">交费金额</td>
      <td width="15%" align="center">是否交清</td>
    </tr>
    <s:set name="stuArray" value="#request.stuArray" scope="action"/>
    <s:if test="#stuArray!=null">
      <s:iterator value="#stuArray" status="status">
        <tr>
          <td width="5%" align="center"><s:property value="#status.count"/></td>
          <td width="12%" align="center"><s:property value="StudentName"/></td>
          <td width="16%" align="center"><s:property value="MatriNo"/></td>
          <td width="12%" align="center">
            <s:property value="@com.csai.db.StudentUtil@haveSplitSpec(SpecialityId)"/>
          </td>
          <td width="15%" align="center">
            <s:hidden name="stuParamArray[%{#status.index}].StudentId"
              value="%{StudentId}"/>
            <s:property value="@com.csai.db.StudentUtil@haveSplitClass(ClassId)"/>
          </td>
          <td width="15%" align="center">
            <s:if test="ClassId!=0">
              <s:textfield name="stuParamArray[%{#status.index}].PayAmount"
                value="%{PayAmount}" size="12"/>
            </s:if>
          </td>
          <td width="15%" align="center">

```



```

        <s:if test="ClassId!=0">
            <s:radio name="stuParamArray[%{#status.index}].PayOK" value="%{PayOK}"
                list="#{1:'是',0:'否'}"/>
        </s:if>
    </td>
</tr>
</s:iterator>
</s:if>
<tr>
    <td width="100%" bgcolor="#C0C0C0" align="center" colspan="8">
        <font color="#0000FF">
            <s:submit value="确定"/>
        </font></td>
</tr>
<s:hidden name="action" value="update"/>
</table>
</s:form>
</body>
</html>

```

这部分的关键代码是如下的 3 个标签：

```

<s:hidden name="stuParamArray[%{#status.index}].StudentId"
    value="%{StudentId}"/>
<s:textfield name="stuParamArray[%{#status.index}].PayAmount"
    value="%{PayAmount}" size="12"/>
<s:radio name="stuParamArray[%{#status.index}].PayOK" value="%{PayOK}"
    list="#{1:'是',0:'否'}"/>

```

如何根据查询的结果动态地在表单中出现文件输入框、单选按钮，提交后又能正确地接收数据并处理，实现的方法与报到分班功能的实现方法相同，此外就不再赘述了，读者可自行阅读源代码或查看“12.4.7 报到分班功能的实现”中的详细解说。



**提示** 设置<s:radio>标签生成的哪个 radio 被选中，是根据<s:radio>标签的 value 属性值来决定的。

本功能的 Action 类代码如下。

#### AcceptMoneyAction.java

```

package com.csai.action;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Map;
import com.csai.POJO.Student;
import com.csai.db.DBConn;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class AcceptMoneyAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String studentname;//学生姓名

```



```

public String action;//操作类型
public String matrino;//录取通知书号
public ArrayList<Student> stuParamArray;//学生数据数组
@Override
public String execute() throws Exception {
    Connection conn=DBConn.createDBConn();
    //----查询数据操作----
    String sqlwhere=new String("");
    String sql=new String("");
    if("select".equals(action)){//如果是查询操作
        if(studentname!=null&&studentname.trim().length()!=0)
            sqlwhere="where studentname like '%" +studentname.trim()+"%' ";
        if(sqlwhere!=null&&sqlwhere.length()!=0){
            if(matrino!=null&&matrino.trim().length()!=0)
                sqlwhere+=" and matrino like '%" +matrino.trim()+"%' ";
        }else{
            if(matrino!=null&&matrino.trim().length()!=0)
                sqlwhere=" where matrino like '%" +matrino.trim()+"%' ";
        }
        sql="select * from student "+sqlwhere;
        Statement state=conn.createStatement();
        ResultSet rs=state.executeQuery(sql);
        ArrayList<Student> stuArray=new ArrayList<Student>();
        while(rs.next()){
            Student stu=new Student();
            stu.setBedchamberId(rs.getInt("bedchamberId"));
            stu.setClassId(rs.getInt("classId"));
            stu.setMatriNo(rs.getString("matriNo"));
            stu.setPayAmount(rs.getFloat("payAmount"));
            stu.setPayOK(rs.getInt("payOK"));
            stu.setRegistDate(rs.getDate("registDate"));
            stu.setSpecialityId(rs.getInt("specialityId"));
            stu.setStudentId(rs.getLong("studentId"));
            stu.setStudentName(rs.getString("studentName"));
            stuArray.add(stu);
        }
        Map request = (Map)ActionContext.getContext().get("request");
        request.put("stuArray", stuArray);
    }
    //----交费操作----
    if(stuParamArray!=null&&"update".equals(action)){
        for(int i=0;i<stuParamArray.size();i++){
            if(stuParamArray.get(i).getClassId()!=0){
                String sqlstr="update student set payamount="+
                    stuParamArray.get(i).getPayAmount()+
                    " payok="+stuParamArray.get(i).getPayOK()+
                    " where studentid="+stuParamArray.get(i).getStudentId();
                Statement state=conn.createStatement();
                state.executeUpdate(sqlstr);
            }
        }
    }
    DBConn.closeConn(conn);
    return SUCCESS;
}

```



```

    }
    /**
     * 此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}

```

要注意，以上操作中仅更新班级 ID 号不为 0 的学生的数据，根据如下的判断：

```
stuParamArray.get(i).getClassId() != 0
```

其实，本质上也就是说已经分过班的才能交费，因为没有交费的学生得到的班级 ID 号为 0，已分班的学生数据则会在班级 ID 号字段中填入班级 ID 号。

### 14.3.8 宿舍分配功能的实现

宿舍分配功能的 JSP 代码如下所示。

bedchamber.jsp

```

<%@ page contentType="text/html; charset=GBK" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<body>
<s:form method="post" action="Bed" theme="simple">
<table border="1" cellpadding="0" cellspacing="0"
    style="border-collapse: collapse"
    bordercolor="#C0C0C0" width="700">
<tr>
<td width="100%" bgcolor="#C0C0C0">
<font color="#0000FF">要查询的条件</font></td>
</tr>
<tr>
<td width="100%">
请输入姓名：
<s:textfield name="studentname"/>
请输入录取通知书号：
<s:textfield name="matrino"/>
<s:hidden name="action" value="select"/>
<s:submit value="提交"/>
</td>
</tr>
</table>
</s:form>
<s:form method="post" action="Bed" theme="simple">
<table border="1" cellpadding="0" cellspacing="0"
    style="border-collapse: collapse"
    bordercolor="#C0C0C0" width="700">
<tr>
<td width="100%" bgcolor="#C0C0C0" align="center" colspan="8">
<font color="#0000FF">查询到的学生数据</font></td>
</tr>
<tr>
<td width="5%" align="center">序号</td>
<td width="12%" align="center">姓名</td>
<td width="16%" align="center">录取通知书号</td>

```



```

        <td width="12%" align="center">录取专业</td>
        <td width="15%" align="center">所在班级</td>
        <td width="15%" align="center">交费金额</td>
        <td width="15%" align="center">是否交清</td>
        <td width="15%" align="center">所在宿舍</td>
    </tr>
    <s:set name="stuArray" value="#request.stuArray" scope="action"/>
    <s:if test="#stuArray!=null">
        <s:iterator value="#stuArray" status="status">
            <tr>
                <td width="5%" align="center"><s:property value="#status.count"/></td>
                <td width="12%" align="center"><s:property value="StudentName"/></td>
                <td width="16%" align="center"><s:property value="MatriNo"/></td>
                <td width="12%" align="center">
                    <s:property value="@com.csai.db.StudentUtil@haveSplitSpec(SpecialityId)"/>
                </td>
                <td width="15%" align="center">
                    <s:hidden name="stuParamArray[%{#status.index}].StudentId"
                        value="%{StudentId}"/>
                    <s:property value="@com.csai.db.StudentUtil@haveSplitClass(ClassId)"/>
                </td>
                <td width="15%" align="center">
                    <s:if test="ClassId!=0">
                        <s:textfield name="stuParamArray[%{#status.index}].PayAmount"
                            value="%{PayAmount}" size="12"/>
                    </s:if>
                </td>
                <td width="15%" align="center">
                    <s:if test="ClassId!=0">
                        <s:radio name="stuParamArray[%{#status.index}].PayOK" value="%{PayOK}"
                            list="#{1:'是',0:'否'}"/>
                    </s:if>
                </td>
                <td width="15%" align="center">
                    <s:if test="ClassId!=0&&PayOK==1">
                        <s:select name="stuParamArray[%{#status.index}].BedchamberId"
                            listKey="bedchamberId"
                            listValue="bedchamberName" list="#request.bedArray"
                            headerKey="0" headerValue=="请选择==" value="BedchamberId">
                    </s:select>
                    </s:if>
                </td>
            </tr>
        </s:iterator>
    </s:if>
    <tr>
        <td width="100%" bgcolor="#C0C0C0" align="center" colspan="8">
            <font color="#0000FF">
                <s:submit value="确定"/>
            </font></td>
    </tr>
    <s:hidden name="action" value="update"/>
</table>
</s:form>
</body>
</html>

```

在显示宿舍下拉框的前面先要作判断：

```
<s:if test="ClassId!=0&&PayOK==1">
```

也就是已经分了班且学费已经付清的学生才能分配宿舍。ClassId!=0 表明已经分了班，PayOK 表示学费已经付清。

此处就不再列出 Action 类的完整源代码了，读者可自行查看随书光盘中的源代码。

## 14.4 小结

读者通过本章的系统开动手操作后，应当对 Struts 2 有了进一步的认识，并能开始在自己的软件开发项目中应用 Struts 2 框架技术。

使用 Struts 2 的优点是显而易见的，简单易用（虽然对初学者还是有一定的难度），它也常和 Hibernate、Spring 等框架集成应用，后续章节将会给出利用 Hibernate 技术来改进本系统的方法。





# 15

## Hibernate 4 持久化技术

Hibernate 其实就是一种 ORM (Object-Relation Mapping, 对象-关系映射) 中间件, 说到底就是可以将数据库表中的数据包装为 Java 对象, 又可以将 Java 对象映射到数据库表中的数据, 于是开发人员可以利用面向对象编程的思想来操作数据库中的数据。如何在 Java 对象与数据库中的数据之间保持一致, 如何在对象与数据库表之间进行映射, 这些工作就交由 Hibernate 来完成, 开发人员就可以专注于实现业务逻辑了。

可见 Hibernate 在 Java 对象与数据库中的关系型数据之间架起了桥梁, 可以将 Java Web 系统封装得更有层次性, 从而具有清晰的系统架构。本章将对 Hibernate 做出精要的介绍, 并辅以实例解说。

### 15.1 Hibernate 介绍

Java 开发人员要操作数据库中的数据往往是通过 JDBC 驱动程序, 使用 SQL 语句来实现的, 这样就需要 Java 开发人员对 Java 编程、数据库的数据结构都有较深的了解, 特别是当要进行事务处理、数据完整性约束等复杂的数据处理时, 对开发人员的要求较高。于是, Hibernate 应运而生了。

#### 15.1.1 Hibernate 的作用

Hibernate 架起了 Java 对象与数据库中的关系型数据的桥梁, 它可以将关系型数据映射成 Java 对象, 将 Java 对象映射成关系型数据, 此种功能又称为 ORM, 可见 Hibernate 就是一种 ORM 中间件。

Hibernate 还可以用来作持久化和反持久化, 持久化是指内存中的对象及对象之间的关系持久化到数据库表所表示的关系, 反持久化则是将数据库中的二维关系示例化到内存中。因此, Hibernate 常位于数据库和应用程序之间, 如图 15-1 所示。

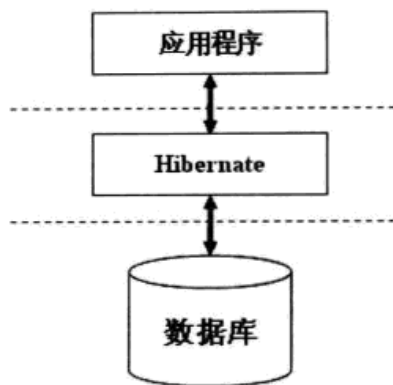


图 15-1 Hibernate 所处的位置

使用 Hibernate 的好处是降低了开发人员的劳动强度，通过 Hibernate 的对象-关系映射，程序员即可免去繁重的编码工作量，只需要在映射文件中对关系进行定义，然后编写少量的代码，便可将实体与关系的维护、对象与关系的转换工作交由 Hibernate 代劳。

### 15.1.2 Hibernate Core for Java

目前最新的 Hibernate 软件版本是 Hibernate 4.0，本书就使用这个版本。Hibernate 4 是一个产品套件，其中最为常用的就是 Hibernate Core for Java，这是 Hibernate 的核心组件，可以为应用程序提供强大、高性能的对象-关系映射以及查询服务。

### 15.1.3 Hibernate 的核心 API

接下来就需要了解 Hibernate 的核心 API，开发人员要熟悉所有的 API 是比较困难的，但是需要理解核心的 API，这样就可以快速进行开发了。主要要理解如下这些：

(1) 数据库操作相关的 API：主要有 Session、Transaction 和 Query，用来完成基本的创建、读取、更新、删除操作以及查询操作。

(2) 配置文件操作 API：主要就是 Configuration 类。

(3) 回调功能 API：它允许应用程序能对一些事件的发生做出相应的操作，主要包括 Interceptor、Lifecycle 和 Validatable。

(4) 扩展功能 API：指一些可以用来扩展 Hibernate 的映射机制的接口，例如 UserType、CompositeUserType 和 IdentifierGenerator 等，这些接口可由用户程序来实现。

通俗地讲，Hibernate 启动时，通过 Configuration 接口获取目前的配置（如数据源、连接用户名及密码、数据库方言以及最大连接数等）并将这些配置加载到内存中，当执行到初始化一个 SessionFactory 接口时，便一次性创建这个线程安全的 Session 工厂，每当需要执行持久化操作时，应用程序就会从这个工厂中取一个 Session，直至其生命周期结束。在持久化的过程中，可能会用到 Transaction 接口进行事务处理，也可能通过 Query 和 Criteria 接口进行查询，Hibernate 底层会自动实现 Callback 接口。



## 15.2 安装与配置 Hibernate 4

### 15.2.1 下载 Hibernate 4

Hibernate 的下载页面如图 15-2 所示。

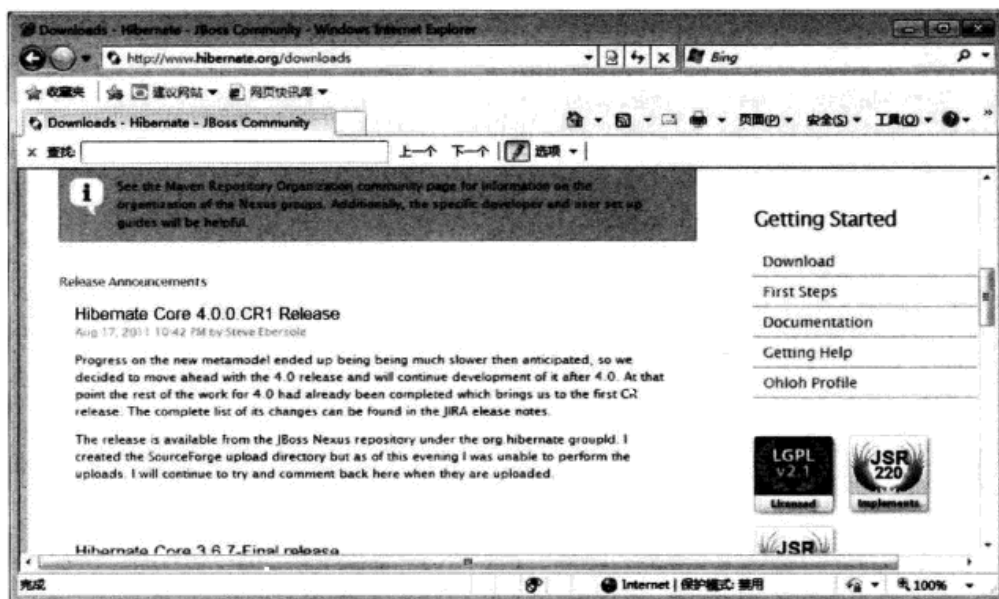


图 15-2 Hibernate 的下载页面

这是“<http://www.hibernate.org>”官方网站的下载页面，从地址栏可以看到下载的页面地址为“<http://www.hibernate.org/downloads>”。图 15-2 中列出了可供下载的 Hibernate 套件中的各个部件。这里暂时仅需下载“Hibernate Core”。



**提示** 读者在打开这个页面时可能和图 15-2 稍有不同，因为可能有版本的变化，只需要找到所需要的组件下载即可。

开发时，需要将 lib\required 目录下的所有 jar 文件复制到正在开发的 Web 应用的“WEB-INF/lib”目录中，即可在 Web 应用中使用 Hibernate 4 了，本书中采用的是 Hibernate 4.0 版本，如图 15-3 所示。

### 15.2.2 配置 Hibernate 4

使用 Hibernate 进行开发，程序员的一项重要工作就是修改配置文件，因此需要了解常用的配置文件，以及配置的选项。

Hibernate 配置文件主要用于配置数据库连接和 Hibernate 运行时所需的各种属性。Hibernate 配置文件允许传统的 hibernate.properties 文件和 hibernate.cfg.xml 文件同时存在。

Hibernate 配置文件的存放位置为类文件的根目录，即 Web 应用的 WEB-INF 目录。Hibernate 初始化时到类文件的根目录下寻找这个文件并读取其中的配置信息，其中 xml 格式文件的优先级高于 properties 格式文件，也就是说如果两个文件同时存在，则 hibernate.cfg.xml 的配置会起作用，而 hibernate.properties 的配置则会被屏蔽。



图 15-3 Hibernate 4.0



**提示** 建议使用 XML 配置文件，因为可读性更强。



**提示** 在 Hibernate 下载后解压缩文件包中的 etc 下有常用的配置文件示例，供开发人员参考。

### 【实例 15-1】hibernate.cfg.xml 的配置文件示例

hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <!--Sessionfactory 配置-->
    <session-factory>
        <!-- 数据库连接设置 -->
        <!--SQL 方言-->
        <property name="dialect">
            org.hibernate.dialect.SQLServerDialect
        </property>
        <property name="connection.username">sa</property>
        <property name="connection.password">123</property>
        <property name="connection.url">
            jdbc:sqlserver://localhost:1433;DatabaseName=testDatabase
        </property>
    </session-factory>
</hibernate-configuration>
```



```

<!--JDBC 连接池 -->
<property name="connection.pool_size">1</property>
<!--是否将运行期生成的 SQL 输出到日志以供调试-->
<property name="show_sql">true</property>
<!--映射资源，配置文件必须包含其相对的全路径-->
<mapping resource="notebook/Notebook.hbm.xml" />
</session-factory>
</hibernate-configuration>

```

<hibernate-configuration>中可以包含多个用于创建 SessionFactory 示例的<session-factory>标记，每一个 SessionFactory 可以连接不同的数据源。建议开发人员在一个配置文件中定义一个 SessionFactory，可以将多个 SessionFactory 分别定义在不同的配置文件中，然后在程序中动态加载多个配置文件（hibernate.cfg.xml 为默认的配置文件中）。

<session-factory>标记中可以定义若干常用的属性标记<property>，具体含义如下：

- (1) hibernate.connection.url：数据库连接字符串；
- (2) hibernate.connection.username：数据库登录用户名；
- (3) hibernate.connection.password：数据库登录密码；
- (4) hibernate.dialect：SQL 方言；
- (5) hibernate.show\_sql：是否在控制台输出 SQL 语句。

在以上 5 个参数中，hibernate.connection.url、hibernate.connection.username、hibernate.connection.password 这 3 个参数容易理解，也就是数据库连接要用到的一些连接参数，但 SQL 方言指的是什么呢？因为不同的数据库使用的 SQL 是稍有差异的，这就需要 Hibernate 来区别对待，这个参数就用于指出使用的是何种数据库的 SQL 语言。常用的数据库语言如下：

- (1) DB2：org.hibernate.dialect.DB2Dialect；
- (2) MySQL：org.hibernate.dialect.MySQLDialect；
- (3) Oracle：org.hibernate.dialect.OracleDialect；
- (4) Sybase：org.hibernate.dialect.SybaseDialect；
- (5) Microsoft SQL Server：org.hibernate.dialect.SybaseDialect；
- (6) Informix：org.hibernate.dialect.InformixDialect。

此外，在<session-factory>中还可以定义<mapping>标记，该标记可以有零到多个，其 source 属性指向实现对象-关系映射的映射文件，这些文件应与持久化类处于同一级目录，也就是说应该位于同一个包内。

## 15.3 一个简单的 Hibernate Web 应用

下面来开发一个简单的 Hibernate Web 应用，以便了解开发的全过程，并快速上手开发。

### 【实例 15-2】一个简单的 Hibernate Web 应用

在 Eclipse 中新建一个动态的 Web 工程 ch15，并进行开发，如图 15-4 所示。

在“src”源代码目录下新建两个包，一个是 hibernate 包，其中仅有一个类 HibernateUtil，

用于生成 Hibernate 会话；另一个是 notebook 包，其中包括对应数据库中的 Notebook 表的 Notebook 类，表示对应关系的 Notebook.hbm.xml 配置文件。



图 15-4 工程 ch15 的情况

WebContent 目录下是所有的 Web 程序，其中“WEB-INF/lib”目录中有所有 Hibernate 的 jar 包及数据库驱动程序 jar 包。此外还需要 slf4j-log4j12-1.5.2.jar 和 log4j-1.2.16.jar 包，其中 slf4j-log4j12-1.5.2.jar 的下载地址如下：

<http://mirrors.ibiblio.org/pub/mirrors/maven2/org/slf4j/slf4j-log4j12/1.5.2/slf4j-log4j12-1.5.2.jar>

slf4j-log4j12-1.5.2.jar 的下载地址如下：

<http://logging.apache.org/log4j/1.2/download.html>

#### ① 在数据库中建表

这里要用到一个 Notebook 表，建表的 SQL 语句如下：

```
CREATE TABLE [Notebook] (
    [NoteId] [varchar] (5) COLLATE Chinese_PRC_CI_AS NOT NULL ,
    [Title] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL ,
    [Content] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL ,
    [Noter] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL ,
    CONSTRAINT [PK_Notebook] PRIMARY KEY CLUSTERED
    (
        [NoteId]
    ) ON [PRIMARY]
) ON [PRIMARY]
```

从以下语句可以看出，Noteld 为主键；Title 表示标题，内容不能为空；Content 表示记载的文本，内容可以为空；Noter 表示记载人，内容不能为空。

#### ② 建立表对应的持久化对象和配置文件

Notebook 表对应的持久化对象是相当简单的，无非就是将数据库表字段对应的字段名作为属性名，再配以 getXxxx()方法和 setXxxx()方法，源代码如下。

##### Notebook.java

```
package notebook;
import java.io.Serializable;
public class Notebook implements Serializable {
    //属性
```



```

private String noteId;
private String title;
private String content;
private String noter;
//构造函数
public Notebook() {
}
public Notebook(String noteId, String title, String noter) {
    this.noteId = noteId;
    this.title = title;
    this.noter = noter;
}
}
/**
    此外省去各个属性的 getXxxx()方法和 setXxxx()方法代码
*/
}

```

相应地持久化配置文件如下。

Notebook.hbm.xml

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
<class
    name="notebook.Notebook"
    table="Notebook" schema="dbo" lazy="false">
    <id
        name="noteId" type="java.lang.String" column="NoteId">
        <generator class="assigned" />
    </id>
    <property name="title" type="java.lang.String"
        column="Title" not-null="true" length="20"/>
    <property name="content" type="java.lang.String"
        column="Content" length="50"/>
    <property name="noter" type="java.lang.String"
        column="Noter" not-null="true" length="20"/>
    </class>
</hibernate-mapping>

```

**<class>**标签的 **name** 属性指出了持久化类的类名，**table** 属性指出数据库中对应的表名，**schema** 属性指出数据库模式名，**lazy** 属性为 **false** 表示预先抓取。

**<id>**标签表示是关键字，**name** 属性指出 Java 类中的名称，**type** 属性指出在 Java 中对应的数据类型，**column** 属性指出数据库表中对应的字段名；**<generator>**标签表示由应用程序来指定值。接下来的其他标签分为表述了其他的 Java 类属性与数据库表字段的对应关系。

### ③ 编写 hibernate.cfg.xml 配置文件

hibernate.cfg.xml

```

<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"

```



```

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="dialect">
      org.hibernate.dialect.SQLServerDialect
    </property>
    <property name="connection.username">sa</property>
    <property name="connection.password">123</property>
    <property name="connection.url">
      jdbc:sqlserver://localhost:1433;DatabaseName=testDatabase
    </property>
    <property name="show_sql">true</property>
    <mapping resource="notebook/Notebook.hbm.xml" />
  </session-factory>
</hibernate-configuration>

```

配置文件中有一个<mapping>标签，表示映射资源，并在 resource 属性中给出了 XML 配置文件相对“WEB-INF/classes”的目录。

#### ④ 编写生成会话的类

##### HibernateUtil.java

```

package hibernate;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    public static final ThreadLocal session = new ThreadLocal();
    static {
        try {
            // 根据配置文件 hibernate.cfg.xml 创建 SessionFactory
            sessionFactory = new Configuration().configure().buildSessionFactory();
            System.out.println("初始化 SessionFactory 成功。");
        } catch (Throwable ex) {
            System.err.println("初始化 SessionFactory 失败。" + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static Session getSession() {
        Session s = (Session) session.get();
        // 获得一个新的 Session
        if (s == null) {
            s = sessionFactory.openSession();
            session.set(s);
        }
        return s;
    }
}

```

这个类中有一个静态的属性 sessionFactory，表示会话工厂。后续还有一段静态的代码，表示第一次使用这个类生成对象时会执行，这段代码中生成了会话工厂的实例。getSession()方法使用会话工厂生成了一个新的会话，并打开。



## ⑤ 编写测试用的 JSP 页面

firstHibernate.jsp

```

<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.util.*,notebook.*,org.hibernate.Session,
                hibernate.*" %>
<html>
<head>
<title>使用 Hibernate</title>
</head>
<body>
<%
Session sessionHiberante=HibernateUtil.getSession();
Notebook noteBook=new Notebook();
noteBook.setNoteId("001");
noteBook.setTitle("第一次记录");
noteBook.setContent("我们开始第一个使用 Hibernate 的程序了。");
noteBook.setNoter("dzy");
sessionHiberante.beginTransaction();      //开始事务
sessionHiberante.save(noteBook); //持久化 noteBook 对象
ArrayList result =(ArrayList)sessionHiberante.createQuery("from Notebook").list();
for (int i=0;i<result.size();i++){
    Notebook notebook=(Notebook)result.get(i);
    out.println(notebook.getTitle()+":"+notebook.getContent()+"<br>");
}
sessionHiberante.getTransaction().commit(); //结束事务
%>
</body>
</html>

```

JSP 页面中先是用 `HibernateUtil` 类得到一个会话；再新建了一个 `Notebook` 对象，并设置各个属性的值；然后在会话中开始事务，往事务中保存 `Notebook` 对象；接下来使用会话的 `createQuery()` 方法做了一个数据查询，查询出所有的 `Notebook` 记录，并返回列表，此后迭代输出列表中的元素；最后提交事务。程序的运行效果如图 15-5 所示。

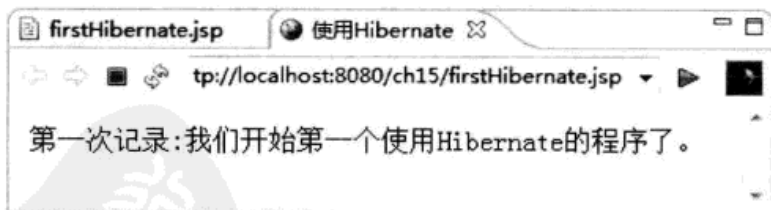


图 15-5 使用 Hibernate 的 JSP 页面运行结果

从图 15-5 可以看出，成功地向数据库表中插入了一条记录，并显示出了记录的 `title` 字段和 `content` 字段的内容。在控制台中有如图 15-6 所示的输出，其中显示了程序运行过程中执行的 SQL 语句。

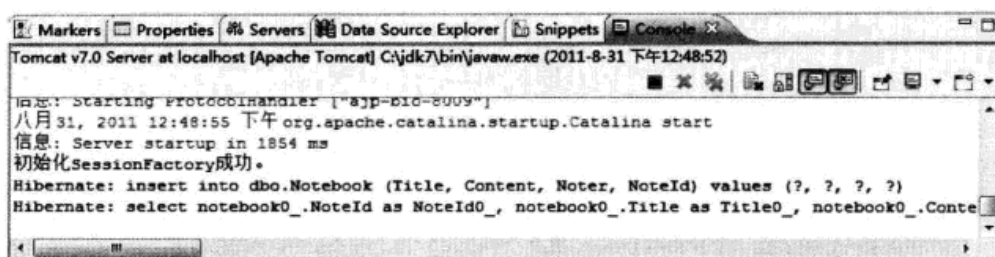


图 15-6 控制台输出

## 15.4 持久化对象

持久化对象无非也就是简单的 POJO 对象，一个与数据库表对应的包含有若干属性，以及属性对应的 `getXxxx()` 方法和 `setXxxx()` 方法的类的实例。这个对象由 Hibernate 来进行管理，存在有各种不同状态，并可以在不同的状态之间进行转换，如图 15-7 所示。

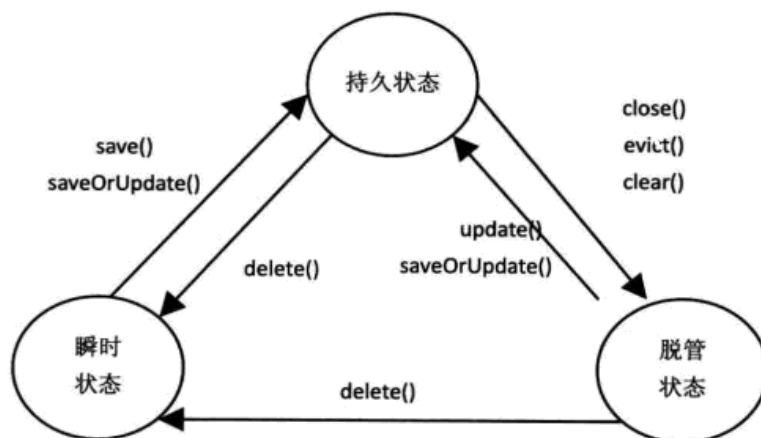


图 15-7 对象的状态及状态之间的转换

瞬时状态对象指已经创建，但尚未与 Hibernate Session 关联的对象，也就是说该对象尚未被 Session 对象的 `save()`、`update()` 等方法持久化到数据库中去，并且未被赋予持久化标示。如果瞬时对象在程序中没有被引用，它将被垃圾回收器销毁。

### 15.4.1 Session 接口

下面介绍 Session 接口的一些常用的方法。

#### 1. beginTransaction()

Transaction `beginTransaction()` throws `HibernateException`

此方法开始事务，并返回事务对象。

#### 2. clear()

清除会话中的所有持久化对象，并将未处理完的对象保存、更新、删除操作。



### 3. close()

关闭会话，此时会释放数据库连接。

### 4. createQuery()

`Query createQuery(String queryString) throws HibernateException`

此方法使用 `queryString` 参数指定的 HQL 语句创建一个查询对象，并返回这个查询对象。通过查询对象 `Query` 的 `list()` 方法可以将查询结果以 `List` 对象返回。

### 5. createSQLQuery()

`SQLQuery createSQLQuery(String queryString) throws HibernateException`

此方法使用 `queryString` 参数指定的 SQL 语句创建一个查询对象，并返回这个查询对象。

### 6. delete()

`void delete(Object object) throws HibernateException`

此方法用于将数据库中与持久对象或脱管对象相对应的记录删除。如果传入的参数是持久对象，`Session` 将执行一个 `delete` 语句。如果传入的参数是脱管对象，则将先把脱管对象重新与 `Session` 建立关联，然后再执行 `delete` 语句。持久对象或脱管对象在被删除之后变为瞬时对象。如果对象的映射配置中 `cascade` 值为 `delete`，则会级联删除相关联的对象与数据。

### 7. evict()

`void evict(Object object) throws HibernateException`

此方法从 `Session` 的缓存中删除持久化对象，使对象变为托管状态。如果对象的映射配置中 `cascade` 值为 `evict`，则会将级联的对象变为托管状态。

### 8. flush()

强制清理会话，调用此方法会将持久状态的对象与数据库数据对应起来。常在事务提交前或会话结束前调用此方法，但并不是一定要调用这个方法，这取决于 `FlushMode` 参数，有如下几种取值：

- ALWAYS：每做一次查询都会清理一次会话，这样做效率较低。
- AUTO：默认值，将为确保查询会得到新的状态而自动做清理处理。
- COMMIT：在每次提交事务时自动作理。
- MANUAL：在应用程序中手动调用 `flush()` 方法时才做清理。

### 9. get()

`Object get(Class clazz, Serializable id) throws HibernateException`

此方法根据指定的标识符返回持久化对象，如果找不到则返回 `null`。

### 10. load()

`Object load(Class theClass, Serializable id) throws HibernateException`

此方法根据指定的标识符返回持久化对象，如果找不到则抛出异常。

### 11. persist()

```
void persist(Object object) throws HibernateException
```

此方法使一个瞬时状态的对象持久化，如果映射配置中设置为 `cascade="persist"`，则将会使相关联的对象也持久化。

### 12. save()

```
Serializable save(Object object) throws HibernateException.
```

此方法持久化一个瞬时状态的对象。

### 13. update()

```
void update(Object object) throws HibernateException
```

此方法使脱管对象持久化。对于持久化对象的修改一般不需要专门调用某个方法而将所做的修改持久化，`Session` 的刷新(`flush`)机制可以自动将持久化对象的数据与数据库同步。`update()` 方法主要用于使脱管对象重新关联到 `Session` 从而将修改内容持久化。

### 14. saveOrUpdate()

```
void saveOrUpdate(Object object) throws HibernateException
```

`saveOrUpdate()` 方法是一个既可自动分配新的持久化标示符来保存瞬时对象，又可更新或者重新关联脱管对象的通用方法。但是，只要没有在某一个 `Session` 中使用来自另外一个 `Session` 的示例，都不必用到 `update()`、`saveOrUpdate()` 等方法。该方法同时具备了 `save()` 方法和 `update()` 方法的功能，并具有自动检测参数类型的功能，如果参数为瞬时对象，则执行 `save()` 方法，如果参数为脱管对象，则执行 `update()` 方法。

## 15.4.2 映射配置

在采用 `Hibernate` 开发的应用系统中一个数据库的表就对应着一个映射配置文件和一个 `JavaBean` 类（实际上就是一个 `POJO` 类）。`JavaBean` 与数据库表的对应关系通过映射配置文件来定义，即 `hbm.xml` 文件。下面将详细解释映射配置文件中的配置项。

### 1. <hibernate-mapping>

`<hibernate-mapping>` 标签是根元素，使用的格式如下：

```
<hibernate-mapping schema="数据库 schema 名称" catalog="数据库 catalog 名称"
    default-cascade="级联风格" default-access="属性访问策略"
    default-lazy="true|false" auto-import="true|false" package="持久化类的包路径">
    被嵌套的其他标签
</hibernate-mapping>
```

`schema` 属性用于定义数据库 `schema` 的名称。某些不支持 `schema` 的数据库，如 `MySQL`，不需要配置该属性；在 `MS SQL Server` 中，`schema` 对应其中的“用户”概念，如 `dbo` 或其他自



定义用户；在 Oracle 中支持 schema 的概念。

catalog 属性用于定义数据库 catalog 的名称。某些不支持 catalog 的数据库，如 Mysql 中，不需要配置该属性；在 MS SQLServer 中，catalog 对应其中的“数据库名”概念；在 Oracle 中支持 catalog 的概念。

default-cascade 属性用于定义默认的级联风格，不使用该属性则采用其默认值 none，即不进行级联操作。允许的取值有：none、save-update、delete、all、delete-orphan、all-delete-orphan 等。如果在<class>元素所包含的子元素（如 many-to-one 或 set）中又重新定义了 cascade 属性，则 Hibernate 将优先执行新的级联操作。如果某子元素没有定义 cascade 属性，Hibernate 将执行 default-cascade 所定义的级联操作。

default-access 属性用于定义属性访问策略，默认值为 property，允许的取值有 property、field、自定义类名。property 表示通过 getXxxx()方法和 setXxxx()方法得到和设置属性的值；field 表示使用反射机制来访问类的属性；自定义的类则需要创建一个 PropertyAccessor 接口的实现类，并将 default-access 属性设为该类的名称。

default-lazy 属性用于定义默认的加载风格，默认值为 true，即当 Hibernate 在执行检索时，不将所有相关联的持久化对象同时加载，仅加载符合检索条件的对象，这称为延迟抓取。如果该属性值设置为 false，则将同时加载所有的关联对象，这称为预先抓取。如果在<class>元素所包含的子元素（如 many-to-one 或 set）中又重新定义了 lazy 属性，则 Hibernate 将优先执行新的加载风格。

auto-import 属性用于设置是否可以在查询语言中使用非全路径的类名，默认值为 true。如果将该属性设置为 Notebook.hbm.xml，那么当读者在编写查询语句时，就可以直接使用类名，如可以用“select \* from Notebook”，而不必使用“select \* from hibernate.chapter15. Notebook”。

package 属性用于设置一个包前缀，如果在映射文件中没有指定全路径的类名，就使用该属性的值作为包名。

## 2. <class>

<class>标签用于定义持久化类，定义的方法如下：

```
<class name="持久化类名" table="数据库表名"
discriminator-value="用于区分不同子类的标志值"
mutable="是否可被更新或删除选项: true|false"
schema="数据库模式名称" catalog="数据库 catalog 名称"
proxy="延迟装载的代理类" dynamic-update="动态更新选项: true|false"
dynamic-insert="动态插入选项: true|false" select-before-update="true|false"
polymorphism="多态选项: implicit|explicit" where="附加的 where 子句"
persister="对象持久化的实现类的类名" batch-size="抓取数量"
optimistic-lock="乐观锁定选项: none|version|dirty|all"
lazy="延迟抓取选项: true|false"/>
```

name 属性用全路径指出持久类的名称，如果在 hibernate-mapping 元素中已经指定了 package 属性的值，则该属性可以只给出类名。该属性的值为一个接口类，则可用<subclass>标签来指定该接口的实际实现类。

table 属性指定了持久化类所对应的数据库表名，如果不对该属性进行设置，则其默认值为

`class` 属性所定义的不包括包路径的类名。`mutable` 属性用于设置该持久化类的示例是否是可变的，默认值为 `true`，即持久化对象是可以被更新或者删除的。如果将该属性设置为 `false`，则该持久化类的示例不可以被应用程序更新或者删除。

`dynamic-update` 属性用于指定 `Update` 语句是否会在运行时动态生成，并且只更新那些修改过的字段，其默认值是 `false`，即不动态生成 `update` 语句，执行更新操作时将更新所有字段。`dynamic-insert` 属性用于指定 `insert` 语句是否会在运行时动态生成，并且只插入那些不能为空的字段，其默认值是 `false`，即不动态生成 `insert` 语句，执行插入操作时将插入持久化对象中有值的字段。



**提示** 当表中字段较多时，建议把 `dynamic-update` 属性和 `dynamic-insert` 属性都设为 `true`，这样在 `insert` 和 `update` 语句中就只包含需要插入或更新的字段，这样可以节省数据库执行 SQL 语句的时间，从而提高应用的运行性能。

`polymorphism` 属性用于指定是 `implicit`（隐式）还是 `explicit`（显式）的使用查询多态，默认值是 `implicit`。`discriminator-value` 属性值默认与 `class` 类名相同，用于区分不同子类的值。

`schema` 属性和 `catalog` 属性均可用于覆盖在根元素 `<hibernate-mapping>` 中指定的相应属性。`select-before-update` 属性设置是否在更新（`update` 操作）前确认值被修改，如果是则需要确认，如果没有修改就不做更新操作，这意味着再更新时要做一次 `select` 操作，该属性的默认值为 `false`。

`where` 属性用于增加一个 `where` 子句，则在每次操作时都会加上这个 `where` 子句。`batch-size` 属性设置批次抓取数量，默认值为 1。`optimistic-lock` 属性用于设置乐观锁定的情况。

### 3. <id>

`id` 元素用于定义持久化类中映射到数据库表主键的属性，使用的方法如下：

```
<id name="属性名" type="类型名" column="字段名"
    unsaved-value="any|none|null|id_value"
    access="属性访问策略: field|property|ClassName">
    <generator class="持久化标识符生成策略"/>
</id>
```

`name` 属性指出持久化类中与数据库主键对应的属性名；`type` 属性指出 Hibernate 数据类型；`column` 属性指出数据库表的主键字段名。

`unsaved-value` 属性用来标识该持久化类的示例是刚刚创建的而且尚未保存，其默认值是 `null`，该属性很少被使用。

`<generator>` 属性用于定义主键的生成策略，默认值为 `assigned`。值为 `increment`，表示递增，可以为 `long`、`short` 或者 `int` 类型生成唯一标识。值为 `identity`，则支持 DB2、MySQL、MS SQL Server、Sybase 和 HypersonicSQL 的内置标识字段，字段数据类型为 `long`、`short` 或者 `int` 类型。值为 `sequence` 表示序列。值为 `assigned` 表示由应用程序来设置值。值为 `foreign` 表示参考其他关联对象的标识符，此时必须和 `<one-to-one>` 联合在一起使用，用在基于主键关联的一对一关联。





**提示** increment 方式与 identity 方式的区别在于：increment 方式是由 Hibernate 实现递增的标识符，每次增量为 1；identity 方式则是依赖于底层数据库的自增主键生成机制，每次的增量可以在数据库中进行控制。



**提示** increment 方式中，Hibernate 在第一次生成主键时先查询数据库表主键的最大值，此后根据最大值来增量式生成主键，但如果是在同一个进程中有删除操作，则基数会持续增长而不受数据删除的影响。

#### 4. <property>

<property> 标签用于定义持久化类属性与数据库表字段之间的对应关系，使用的方法如下：

```
<property name="属性名称" column="数据库表字段的名称" type="类型"
    update="是否可更新: true|false" insert="是否可插入: true|false"
    formula="属性值计算表达式" access="属性值访问策略: field|property|ClassName"
    lazy="是否使用延迟加载: true|false" unique="属性值是否唯一: true|false"
    not-null="是否非空: true|false" optimistic-lock="是否采用乐观锁: true|false"
    generated="是否由数据库生成: never|insert|always"/>
```

name 属性必须给出。column 属性值如果没有给出则默认值与 name 属性值相同。update 属性表示持久化类是否可以被更新，默认值为 true。insert 属性表示持久化类是否可以被插入，默认值为 true。formula 属性给出一个 SQL 表达式，根据这个表达式来计算出持久化类属性的值。unique 属性表示持久化类的属性值是否必须唯一。not-null 属性表示持久化类的属性值是否不能为空。

Hibernate 的映射文件还可以包括 <discriminator>、<many-to-one>、<many-to-many>、<component>、<subclass>、<join>、<version> 等元素，用于实现关联关系映射、继承关系映射、组件映射、事务控制等。

#### 5. 映射数据类型

Hibernate 就像是 Java 应用程序和数据库之间的中介，然而 Java 语言中定义的数据类型与数据库中的数据类型是不一致的，为了解决这个问题，Hibernate 对数据类型做了定义，对应关系如表 15-1 所示。

表 15-1 数据类型的对应关系（以 SQL Server 为例）

序 号	Hibernte 数据类型	数据库数据类型	Java 数据类型
1	integer	int	int、java.lang.Integer
2	yes_no	char	boolean、java.lang.Boolean
3	big_decimal	numeric	java.math.BigDecimal
4	big_integer	numeric	java.math.BigInteger
5	string	varchar	java.lang.String
6	date	datetime、timestamp	java.util.Date
7	time	datetime、timestamp	java.util.Date

续表

序 号	Hibernate 数据类型	数据库数据类型	Java 数据类型
8	Timestamp	datetime、timestamp	java.util.Date
9	calendar	datetime、timestamp	java.util.Calendar
10	calendar_date	datetime、timestamp	java.util.Calendar
11	class	varchar (类全路径)	java.lang.Class
12	locale	varchar (ISO 代码)	java.util.Locale
13	timezone	varchar (ID)	java.util.TimeZone
14	currency	varchar (ISO 代码)	java.util.Currency
15	binary	binary	byte[]
16	text	text	java.lang.String
17	clob	clob	java.sql.Clob
18	blob	blob	java.sql.Blob

## 【实例 15-3】映射配置示例

在本例中，将新建两个数据库表：**student** 表（学生表）和 **class** 表（班级表），并做出映射配置。数据库表的建表语句如下：

```

----班级表----
CREATE TABLE [class] (
    [classId] [bigint] IDENTITY (1, 1) NOT NULL ,
    [className] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL ,
    [addTime] [datetime] NULL CONSTRAINT [DF_class_addTime] DEFAULT (getdate())
) ON [PRIMARY]
----学生表----
CREATE TABLE [student] (
    [studentId] [bigint] NOT NULL ,
    [studentName] [char] (10) COLLATE Chinese_PRC_CI_AS NULL ,
    [classId] [bigint] NULL ,
    [addTime] [datetime] NULL ,
    CONSTRAINT [PK_student] PRIMARY KEY CLUSTERED
    (
        [studentId]
    ) ON [PRIMARY]
) ON [PRIMARY]

```

从表的定义来看，班级表的 **ClassId** 字段由数据库自动生成；学生表的 **bigint** 虽为主键，但数据库并不自动生成。接下来看两个表对应的持久化类。

## Student.java

```

package school;
import java.util.Date;
public class Student {
    public long studentId;
    public String studentName;
    public Date addTime;
    public long classId;
    /**
     此外省去各个属性的 getXxxx() 方法和 setXxxx() 方法代码
    */
}

```



## ClassDefine.java

```

package school;
import java.util.Date;
public class ClassDefine {
    public long classId;
    public String className;
    public Date addTime;
    /**
     此外省去各个属性的 getXxxx() 方法和 setXxxx() 方法代码
    */
}

```

持久化类与数据库表是一一对应的，无非就是属性与对应的 `getXxxx()` 方法、`setXxxx()` 方法。其中 `addTime` 这个属性 `java.util.Date` 数据类型。为什么班级表对应的持久化类要取名为 `ClassDefine`，而不是 `Class` 呢？这是因为 `class` 是 Java 里的关键字，不能再作类名。

再一起来看两个映射文件。

## Student.hbm.xml

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
<class name="school.Student"
    table="student" schema="dbo">
    <id name="studentId" type="java.lang.Long" column="studentId">
        <generator class="increment" />
    </id>
    <property name="studentName" type="string" column="studentName"/>
    <property name="classId" type="java.lang.Long" column="classId"/>
    <property name="addTime" type="timestamp" column="addTime"/>
</class>
</hibernate-mapping>

```

其中 `student` 字段的值由 Hibernate 增量产生，这与班级表的 `ClassId` 字段不同，班级表的 `ClassId` 由数据库系统生成，因此在映射文件中也有所区别。

## ClassDefine.hbm.xml

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
<class name="school.ClassDefine"
    table="class" schema="dbo" dynamic-insert="true">
    <id name="classId" type="java.lang.Long" column="classId">
        <generator class="identity" />
    </id>

```



```

        <property name="className" type="string" column="className"/>
        <property name="addTime" type="timestamp" column="addTime" />
    </class>
</hibernate-mapping>

```

可以看到<generator>标签中的 class 属性值设为 identity，表示值由数据库系统生成。接着修改 hibernate.cfg.xml，加入的两行如下：

```

<mapping resource="school/Student.hbm.xml" />
<mapping resource="school/ClassDefine.hbm.xml" />

```

最后编写一个 JSP 页面来调试程序。

#### school1.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ page import="java.util.*, school.*, org.hibernate.Session,
                hibernate.*" %>

<html>
<head>
<title>使用 Hibernate</title>
</head>
<body>
<%
Session sessionHiberante=HibernateUtil.getSession();
//生成学生对象
Student stu1=new Student();
stu1.setStudentName("曾明军");
stu1.setAddTime(new Date());
stu1.setClassId(1);
Student stu2=new Student();
stu2.setStudentName("王大勇");
stu2.setAddTime(new Date());
stu2.setClassId(1);
//生成班级对象
ClassDefine classObject=new ClassDefine();
classObject.setClassName("软件工程 01 班");
classObject.setAddTime(new Date());
sessionHiberante.beginTransaction(); //开始事务
sessionHiberante.save(stu1); //持久化对象
sessionHiberante.save(stu2);
sessionHiberante.save(classObject);
ArrayList resultStu=(ArrayList)sessionHiberante.createQuery("from Student").list();
ArrayList resultCla=(ArrayList)sessionHiberante.createQuery(
    "from ClassDefine").list();
request.setAttribute("resultStu",resultStu);
request.setAttribute("resultCla",resultCla);
%>
<c:forEach items="${requestScope.resultStu}" var="student">
    <c:out value="${student.studentId}"/>
    <c:out value="${student.studentName}"/>
    <c:out value="${student.classId}"/>

```



```

        <fmt:formatDate value="${student.addTime}"
            pattern="yyyy'年'M'月'd'日' a hh'时'mm'分'"/>
        <br>
    </c:forEach><hr>
    <c:forEach items="${requestScope.resultCla}" var="classDefine">
        <c:out value="${classDefine.classId}"/>
        <c:out value="${classDefine.className}"/>
        <fmt:formatDate value="${classDefine.addTime}"
            pattern="yyyy'年'M'月'd'日' a hh'时'mm'分'"/>
        <br>
    </c:forEach>
    <%
        sessionHiberante.getTransaction().commit();    //结束事务
    %>
</body>
</html>

```

程序中定义了两个 Student 对象，并对属性赋初值，其中 StudentId 没有赋值，因为其值由 Hibernate 自动生成；定义了一个 ClassDefine 对象，并为 ClassName 属性和 AddTime 属性赋了值。程序运行结果如图 15-8 所示。

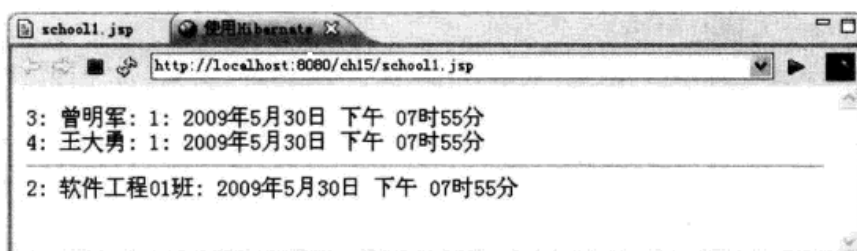


图 15-8 程序运行结果

控制台的输出情况如图 15-9 所示。

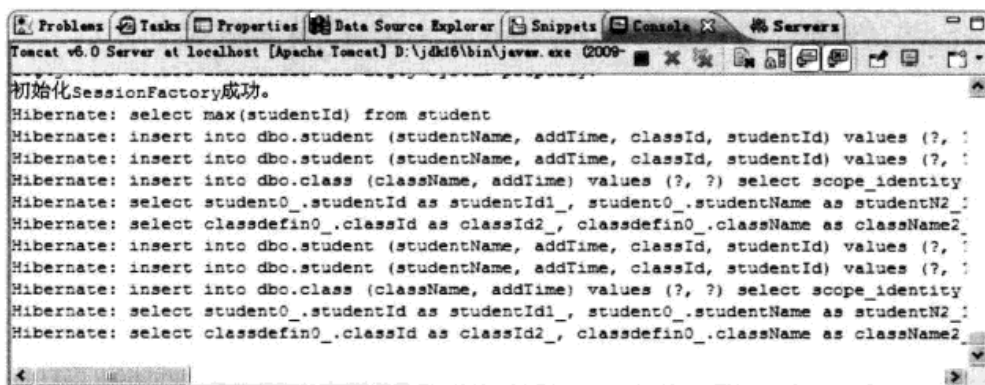


图 15-9 控制的输出情况

从控制台的输出情况来看，做了 6 句查询。第 1 句 select 语句用于查找到 student 表中的 ID 号最大值；第 2、3、4 句直接作插入操作；第 5、6 句将两个表的数据查询出来了。

从图 15-8 所示的情况来看，increment 模式下的学生表如果其中已没有数据，则新的 StudentId 值从 1 重新开始；如果采用 identity 方式，ID 号会不断增长，由数据库系统生成。

## 15.5 关联关系映射

关联关系指的是多对一、一对多、多对多这三种关系，而单向关联指的是从一个持久化对象导航到相关联的另一个持久化对象；双向关联则是指从相关联的任意一方均可导航到另一方，下面详细解决这些关系。

### 15.5.1 单向多对一关联

持久化对象之间的关系就像数据库中表与表之间的关系，然而在关系型数据库理论中，“多对一”关联同于“一对多”关联，且为了消除数据冗余，在两个关系之间不存在“多对多”的关联，“多对多”关联一般需要通过连接表来实现。因此在关系型数据库中只有“一对一”和“一对多”（“多对一”）两种类型的关联，且都是单向的。而在 Hibernate 当中，为了保证关联双方的映射可以通过多种方式进行，“单向一对多”关联与“单向多对一”关联被认为是两种不同的关联，其主要区别在于在哪个表的映射文件中进行<many to one>的配置，进行<many to one>配置的一方便是“多”。

在 Hibernate 的配置文件中使用<many-to-one>标记来定义“多对一”关联。使用方法如下：

```
<many-to-one name="many 方所包含的 one 方所对应的属性名"
column="one 方的主键" class="one 方对应的持久化类的名称"
not-null="true|false" cascade="级联操作选项"/>
```

<many-to-one>标签位于 many 方的配置中，其中 name 属性指出了 many 方所包含的对应 one 方的属性名称；column 属性指出了 one 方的主键；class 属性指出了 one 方对应的持久化类的名称；not-null 属性指出是否不能为空；cascade 属性指出是否会级联操作。

在两个持久化对象之间建立起“多对一”关系要注意以下的问题：

- (1) 在 many 方的映射文件中使用<many-to-one>标记进行关联映射的定义，而非 one 方。
- (2) many 方的持久化类中必须声明与 one 方对应的持久化类的属性，用于实现多对一的导航。
- (3) 级联操作定义在<many-to-one>操作中，如果将 cascade 属性设为 all 或者 save-update，那么在应用程序中只需要用 Session 的 save()方法持久化 many 方的对象即可。

#### 【实例 15-4】多对一关联程序示例

在实例 15-3 的示例中并没有对学生和班级建立数据关系，实际上学生和班级是一个多对一的关系，即一个班可以有多个学生，但一个学生只能属于一个班。下面来看如何建立起这种关系。

修改 Student.hbm.xml，修改后的内容如下。

Student.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
```



```

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
<class name="school.Student"
    table="student" schema="dbo">
    <id name="studentId" type="java.lang.Long" column="studentId">
        <generator class="increment" />
    </id>
    <property name="studentName" type="string" column="studentName"/>
    <property name="addTime" type="timestamp" column="addTime"/>
    <many-to-one name="classDefine" column="classId"
        class="school.ClassDefine" not-null="true"/>
</class>
</hibernate-mapping>

```

原有的 ClassId 属性就删除了，取而代之为<many-to-one>，名称为 ClassDefine，这是 Student 类中的属性，对应着 ClassDefine 类中的 ClassId 属性作为外键。

对于多对一的情况，many 方的类需要修改，one 方的类则不需要修改（配置文件也不需要修改），下面修改 Student 类的代码，修改后的代码如下。

#### Student.java

```

package school;
import java.util.Date;
public class Student {
    public long studentId;
    public String studentName;
    public Date addTime;
    public ClassDefine classDefine;
    /**
     此外省去各个属性的 getXxxx() 方法和 setXxxx() 方法代码
    */
}

```

从代码来看，增加了 ClassDefine 属性，并配有相应的 getXxxx() 方法和 setXxxx() 方法。再编写一个 JSP 页面来调试程序。

#### school2.jsp

```

<%@ page contentType="text/html; charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ page import="java.util.*, school.*, org.hibernate.Session,
    hibernate.*" %>

<html>
<head>
<title>使用 Hibernate</title>
</head>
<body>
<%
Session sessionHiberante=HibernateUtil.getSession();
sessionHiberante.beginTransaction(); //开始事务
//生成班级对象
ClassDefine classObject=new ClassDefine();

```



```

classObject.setClassName("软件工程 02 班");
classObject.setAddTime(new Date());
sessionHiberante.save(classObject);
//生成学生对象
Student stu1=new Student();
stu1.setStudentName("曾向天");
stu1.setAddTime(new Date());
stu1.setClassDefine(classObject);
Student stu2=new Student();
stu2.setStudentName("王月豪");
stu2.setAddTime(new Date());
stu2.setClassDefine(classObject);
sessionHiberante.save(stu1); //持久化对象
sessionHiberante.save(stu2);
ArrayList resultStu =(ArrayList)sessionHiberante.createQuery("from Student").list();
request.setAttribute("resultStu",resultStu);
sessionHiberante.getTransaction().commit(); //结束事务
%>
<c:forEach items="${requestScope.resultStu}" var="student">
    <c:out value="${student.studentId}:"/>
    <c:out value="${student.studentName}:"/>
    <c:out value="${student.classDefine.className}:"/>
    <fmt:formatDate value="${student.addTime}"
        pattern="yyyy'年'M'月'd'日' a hh'时'mm'分'"/>
    <br>
</c:forEach>
</body>
</html>

```

程序运行的结果如图 15-10 所示。

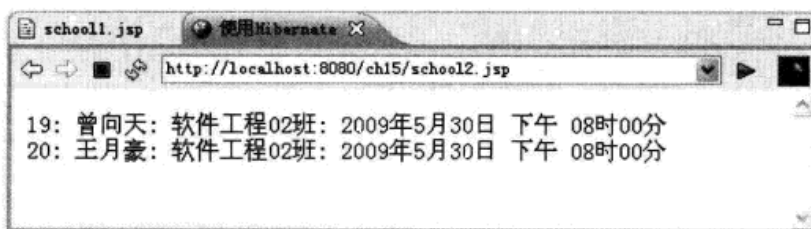


图 15-10 JSP 页面程序调试结果

可见，从中取出了班级名称是通过以下 EL 表达式得到的：

```
${student.classDefine.className}
```

实际上是根据 Student 持久化类的 ClassDefine 属性导航得来的，也就是说从 many 方的对象就直接可以得到 one 方的对象，这样免去了用 SQL 语句时的多次手工查询，也不必做复杂的联合查询，但是其底层原理还是通过 SQL 查询语句来实现的。来看本例执行过的 SQL 语句。

```

Hibernate: insert into dbo.class (className, addTime) values (?, ?)
          select scope_identity()
Hibernate: select max(studentId) from student
Hibernate: insert into dbo.student (studentName, addTime, classId, studentId)
          values (?, ?, ?, ?)

```



```

Hibernate: insert into dbo.student (studentName, addTime, classId, studentId)
values (?, ?, ?, ?)
Hibernate: select student0_.studentId as studentId1_, student0_.studentName as
studentN2_1_, student0_.addTime as addTime1_, student0_.classId as classId1_
from dbo.student student0_

```

第 1 句 SQL 语句是在持久化 classObject 对象时执行的；第 2、3、4 句 SQL 语句是在持久化 stu1、stu2 对象时执行的；第 5 句是在查询数据时执行的，实际上做了联合查询。

在 school2.jsp 页面中，必须先持久化 ClassObject 对象再持久化 stu1、stu2 对象，也就是先持久化 one 方的对象再持久化 many 方的对象，否则会抛出以下的异常：

```

org.hibernate.TransientObjectException: object references an unsaved transient
instance - save the transient instance before flushing: school.ClassDefine

```

表示瞬时状态对象尚未持久化。

## 15.5.2 双向多对一关联

双向多对一经常要用到集合类。JDK 中的集合类分为三种：

(1) Set：集合接口，其中的元素不允许重复。常用的有 HashSet，即使用散列表存储的集合。此种对象与 Iterator 接口结合使用，使用循环指示器循环读取 Set 集合中的元素。

(2) List：列表接口，其中的元素可以重复。常用的有 ArrayList，表示可随需要增长的动态数组；LinkedList，表示链接列表数据结构。此种对象使用索引值顺序读取 List 集合中的元素。

(3) Map：映射接口，每个元素由一个键值对 (key-value) 组成，键对象不可以重复，但值对象可以重复。常用的有 HashMap，即使用散列表存储的映射。此种对象首先使用循环指示器获取 Map 的键集合中的键对象，然后通过键对象读取值对象。

使用双向的多对一要注意以下问题：

(1) 在 many 方的映射文件中使用 <many-to-one> 标记进行关联映射的定义，在 one 方的映射文件中使用集合类映射标记（如 <set>）和 <one-to-many> 标记进行关联映射的定义。

(2) many 和 one 双方都必须声明能够导航到对方的属性，一般在 many 方声明以 one 方持久化类为类型的属性，而在 one 方则声明集合属性（如 java.util.Set），并应用集合类的实现类将该属性初始化。

### 【实例 15-5】双向多对一关联程序示例

在实例 15-4 中实现了学生到班级的多对一关联，通过学生对象即可导航到班级对象，但如何在班级对象中导航到学生对象呢？其实这就是一个双向多对一关系的问题，但班级对应着多个学生，因此也是一个一对多的关联，也就是说学生到班级是多对一关系，班级到学生是一对多联系，设置这两种关联后就形成了双向关联。

先来看学生对象的有关配置。

Student.hbm.xml

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

```

```

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
<class name="school.Student" table="student" schema="dbo">
  <id name="studentId" type="java.lang.Long" column="studentId">
    <generator class="increment" />
  </id>
  <property name="studentName" type="string" column="studentName"/>
  <property name="addTime" type="timestamp" column="addTime"/>
  <many-to-one name="classDefine" column="classId" class="school.ClassDefine"/>
</class>
</hibernate-mapping>

```

从这个配置文件来看，Student 类有 4 个属性，id 属性由 Hibernate 自动生成值，表示学生的 ID 号；name 属性表示学生的姓名；AddTime 属性表示录入时间；ClassDefine 属性表示学生所在的班级对象，其中 ClassId 为外键，相关联的类为 ClassDefine。

#### ClassDefine.hbm.xml

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
<class name="school.ClassDefine" table="class" schema="dto" dynamic-insert="true">
  <id name="classId" type="java.lang.Long" column="classId">
    <generator class="identity" />
  </id>
  <property name="className" type="string" column="className"/>
  <property name="addTime" type="timestamp" column="addTime" />
  <set name="students" inverse="true" cascade="all">
    <key column="classId"/>
    <one-to-many class="school.Student"/>
  </set>
</class>
</hibernate-mapping>

```

从配置文件来看，ClassDefine 类有一个属性用于导航到 Student，inverse="true"表示关联关系由 Student 来维护，cascade="all"表示进行级联更新，这样如果对 ClassDefine 进行操作就会级联影响到 Student，<key column="classId"/>表示主键为 ClassId，<one-to-many class="school.Student"/>表示一对多关系的 many 方为 Student。ClassDefine 类的代码如下。

#### ClassDefine.java

```

package school;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;
public class ClassDefine {
  public long classId;
  public String className;
  public Date addTime;
  public Set students=new HashSet();
  /**
   此外省去各个属性的 getXxxx()方法和 setXxxx()方法代码
  */
}

```



可见里面增加了属性 `students` 的说明, 并配有 `getXxx()` 方法和 `setXxx()` 方法, 属性的数据类型为 `Set`, 需要注意的是数据类型为 `Set`, `Set` 是一个接口, 但新建对象时要使用 `HashSet`, `HashSet` 是 `Set` 接口的实现类。



**提示** `Student` 类的源代码和实例 15-4 中的 `Student` 类是一样的, 在此就不再重复列出了。

下面编写一个 JSP 页面来调试。

#### school3.jsp

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ page import="java.util.*,school.*,org.hibernate.Session,
                hibernate.*" %>

<html>
<head>
<title>使用 Hibernate</title>
</head>
<body>
<%
Session sessionHiberante=HibernateUtil.getSession();
sessionHiberante.beginTransaction();      //开始事务
//生成班级对象
ClassDefine classObject=new ClassDefine();
classObject.setClassName("软件工程 03 班");
classObject.setAddTime(new Date());
//生成学生对象
Student stu1=new Student();
stu1.setStudentName("刘小波");
stu1.setAddTime(new Date());
stu1.setClassDefine(classObject);
Student stu2=new Student();
stu2.setStudentName("邓佳容");
stu2.setAddTime(new Date());
stu2.setClassDefine(classObject);
classObject.getStudents().add(stu1);
classObject.getStudents().add(stu2);
sessionHiberante.save(classObject); //持久化对象
//sessionHiberante.save(stu1);
//sessionHiberante.save(stu2);
ArrayList resultClass =(ArrayList)sessionHiberante.createQuery(
    "from ClassDefine").list();
request.setAttribute("resultClass",resultClass);
sessionHiberante.getTransaction().commit(); //结束事务
%>
<c:forEach items="${requestScope.resultClass}" var="classDefine">
    <c:out value="${classDefine.className}:"/><br>
```

```
<jsp:useBean id="students" class="org.hibernate.collection.PersistentSet"/>
<c:set var="students" value="{classDefine.students}"/>
<c:forEach items="{students}" var="student">
    <c:out value="{student.studentName}:"/>
    <fmt:formatDate value="{student.addTime}"
        pattern="yyyy'年'M'月'd'日' a hh'时'mm'分'"/>
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
</c:forEach>
<br>
</c:forEach>
</body>
</html>
```

这里要注意的地方有三处:

(1) 建立关联关系时要是双向的, 建立关联关系时对象可以是瞬时状态, 代码如下:

```
//建立 stu1 对象到 classObject 对象的关联
stu1.setClassDefine(classObject);
//建立 stu2 对象到 classObject 对象的关联
stu2.setClassDefine(classObject);
//建立 classObject 对象到 stu1 对象、stu2 对象的关联
classObject.getStudents().add(stu1);
classObject.getStudents().add(stu2);
```

(2) 持久化对象时保持级联一方 (classObject 对象) 即可, 由于在配置文件中设置 `cascade="all"`, Hibernate 会自动级联持久化被级联 (stu1 对象、stu2 对象) 的一方。代码如下:

```
sessionHiberante.save(classObject)
```

(3) 用<c:forEach>标签迭代 ClassObject 对象中的 students 属性，由于得到的是一个 Set 接口，需要先行转换为 Set 实现类 HashSet，代码如下：

```
<jsp:useBean id="students" class="org.hibernate.collection.PersistentSet"/>
<c:set var="students" value="${classDefine.students}"/>
```

school3.jsp 页面的运行结果如图 15-11 所示。

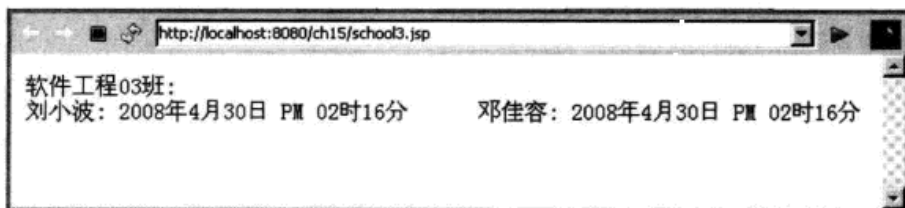


图 15-11 调试双向多对一关系程序的结果

### 15.5.3 一对一关联

一对一关联包括两种形式:

(1) 基于外键的一对一关联: 一个表定义了一个或多个外键, 这些外键参照另一个表的主键。

(2) 基于主键的一对一关联：一个表的主键同时也是外键，该外键参照另一个表的主键，也就是说两个表共享主键。



开发的方法与多对一关系的方法大体相同，只需要注意以下的问题：

(1) 基于外键的一对一关联需要与“多对一”关联配置大致相同，但需要设置一个 `<many-to-one>` 标签表示一个用于导航到 one 方的属性，并将 `<many-to-one>` 标签 `unique` 属性值设置为 `true`，表示只能有一条记录与之关联，即实现了一对一关联。在定义基于外键的一对一关联时，可将 `<many-to-one>` 标签定义在任意一方，同时要在定义 `<many-to-one>` 标签的这一方所对应的持久化类中添加导航到另一方的声明。

(2) 基于主键的“单向一对一”关联的特别之处在于两个表之间共享主键的机制，是通过将一个表的主键同时作为外键参照另一个表的主键而实现的；可将 `<one-to-one>` 标签定义在任意一方，同时要在定义 `<one-to-one>` 标签的这一方所对应的持久化类中添加导航到另一方的声明，另外这一方的持久化标识符必须使用外键标示符生成策略；`<one-to-one>` 标签中应该定义 `constrained="true"`，以约束一个表的主键同时作为外键参照另一个表的主键。

### 15.5.4 多对多关联

多对多关联也是一种极为常见的关联关系。在关系型数据库理论中，两个实体之间如果存在多对多的关系，在数据库中通常使用一个专门的表来表示这种关联，这样做的目的是消除数据冗余。一般的做法是在专门的这张表中放置被关联的两个表的主键。

Hibernate 使用 `<many-to-many>` 标记来定义“多对多”关联。比如学生 `Student` 可以选多门功课 `Lesson` 进行学习，一门功课也可以有多个学生学习，在 `Student` 方的配置文件中就可以增加如下的示例配置：

```
<!--name 属性表示映射属性名，table 属性表示映射到的关联表名-->
<set name="lessons" table="studentLesson" cascade="all">
    <key column="studentId"/>          <!--关联字段名-->
    <!-- 映射到 lesson 表的 lessonId 字段-->
    <many-to-many column="lessonId" class="school.Lesson" />
</set>
```

如果是要建立起双向的多对多关联关系，则关联双方都需要进行 `<many-to-many>` 的定义，关联双方之一的 `<set>`（或其他集合元素）的 `inverse` 属性需要设为 `true`，这样就将主控权交给对方持久化类。

## 15.6 HQL 语言

HQL (Hibernate Query Language) 是 Hibernate 专用的面向对象的查询语言，语法非常类似于 SQL，因此对 SQL 熟悉的读者将可以很快地上手并书写 HQL 语句。HQL 语句书写常用的方法如下：

```
[select|update|delete] [from 类名列表] [where 子句]
[group by 子句] [order by 子句]
```

`[select|update|delete]` 表示查询、更新或删除操作；`from` 子句指出查找的持久化类范围；`where` 子句给出查询的一些条件；`group by` 子句用于对数据进行分组；`order by` 子句用于对查

询的数据进行排序。

### 15.6.1 select|update|delete

**select** 关键词后可以给出要返回的对象或者对象的属性，但是给出的属性必须属于 **from** 子句中列出的类名列表。也可以选择持久化类属性包含的属性，可以直接存入一个 **List** 对象，甚至直接封装成对象，如：

```
//查询出学生成绩中英语课程的成绩
select stu.cores.english from Student as stu
//查询出学生的学号、姓名，并放入一个 List 对象中
select new list(stu.studentNo,stu.studentName) from Student as stu
//查询出学生的学号、姓名，并生成 StuClass 类的实例
select new StuClass(stu.studentNo,stu.studentName) from Student as stu
直接生成 StuClass 类的实例的前提是 StuClass 必须有相应的构造函数，比如：
StuClass(String s1,String s2)
```

如果是做查询也可以不要 **select**，并不给出要返回的对象或者对象的属性，此时会返回 **from** 后指出的类的所有实例。**select** 后还可以使用一些聚合函数，如：**avg()**、**sum()**、**min()**、**max()**、**count()**，此外还可以使用 **distinct**、**all** 等关键词。

**update** 和 **delete** 是 **Hibernate 3** 的新特性，**Hibernate 2** 以前的版本并不支持。也就是说在 **Hibernate 3** 中实现更新数据有两种方法，一种是先修改持久化对象的值；二是使用 **HQL** 语句，例如如下的语句：

```
Transaction tran=session.beginTransaction();
String hql = "delete Student where age>16 ";
Query query = session.createQuery(hql);
query.executeUpdate();
Tran.commit();
```

此语句实现了删除 16 岁以上学生资料的功能。

### 15.6.2 where 子句

**where** 子句可用于有条件地筛选数据。如：

```
from Student as stu where stu.name='李大名'
```

功能是查询出姓名为“李大名”的用户记录。

在 **where** 子句中，读者可以用比较运算符来指定筛选条件，常用的有：**=**、**<>**、**>**、**<**、**>=**、**<=**、**between**、**not between**、**in**、**not in**、**is**、**like** 等。

还可以在 **where** 子句中用算术表达式，如：

```
from Student as stu where (stu.age % 2 = 0)
```

则返回所有年龄为偶数的用户记录。与 **SQL** 相同，可以用逻辑运算符 **not**、**and**、**or** 等来组合逻辑表达式，如：

```
from Student as stu where stu.age>28 and stu.name like '%李%'
```



### 15.6.3 order by 子句

order by 子句用来实现对查询的结果集排序，如：

```
from Student as stu where stu.age>28 order by stu.age
```

在默认情况下，查询结果以升序排列，以上语句就将所得到的结果以学生的年龄升序排列。若要以降序排列，则在后面加一个关键字 desc 即可。如果要根据多个属性排序，则属性之间以“,”分隔，先按前一个排序，值相同的情况下再按后一个排序。

### 15.6.4 group by 子句

group by 子句的作用是将查询结果集按指定的属性值进行分组排列，属性值相同的分在一组。group by 子句后面可跟多个属性名，此种情况下，先按第一个属性值分组，再按第二个属性值在组中分组，原则上可以一直分下去，直到 group by 子句所指的属性值都具有相同值的基本组。

## 15.7 Struts 2 与 Hibernate 4 的集成

Struts 的长处在于实现了 MVC 模式，在用户交互式系统中，可以对数据检验、处理流程进行很好的控制，但是并没有对数据操作进行封装，仍然需要依靠 JDBC 驱动或其他数据库驱动程序来连接数据库；Hibernate 的长处在于 O-R 映射，这样操作数据库中数据的方式改为操作 Java 对象，相比更加符合 Java 编程的习惯，更易于实现业务逻辑与数据操作逻辑。

### 15.7.1 集成的策略

从 Struts 章节的学习内容，以及用 Struts 改进通用在线文章系统的情况来看，开发人员也可能会有这种感觉：Action 类的 execute() 方法中代码太多了，而且有许多重复的，能否复用？能否简化？

在 Action 类的 execute() 方法中，开发人员要想办法得到需要在目标 JSP 页面中显示的所有数据，并进行数据处理，也就是说其中包括了如下的内容：

- (1) 系统的业务逻辑，比如在其中要完成什么样的功能，完成这个功能要进行什么样的操作。
- (2) 数据处理逻辑，包括生成 SQL 语句，生成数据库连接，执行 SQL 查询，处理 SQL 查询结果，甚至还包括数据校验的功能。

有了 Hibernate 以后，就可以对 Action 类的工作继续分层，从而简化工作，提高代码的可复用程度。进行 Struts 与 Hibernate 的集成，如图 15-12 所示。

从图中来看，对数据库的操作工作都由 Hibernate 来完成，这就好比有一个专家对数据库系统的操作进行随时处理，因此工作在 Struts 层的开发人员可以专注于业务逻辑的实现，从而不必太多地关心数据处理的问题。

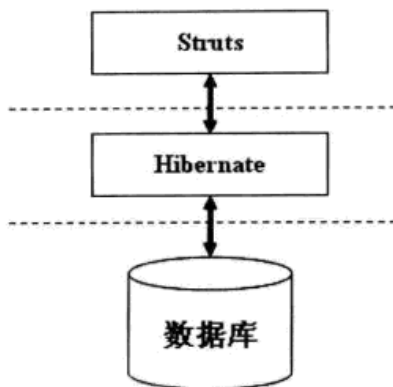


图 15-12 Struts 与 Hibernate 的集成

Struts 与 Hibernate 的整合策略有两种。一种是直接将 Hibernate 的 POJO 对象作为 Struts 中的 Action，在 POJO 对象中再加入 `execute()` 之类的具有 Struts 特性的方法，这种方法虽然程序简单了，但并不提倡。因为 POJO 对象承载的任务过多，既要作为 Action 类来执行业务逻辑，也要作为 POJO 对象来持久化对象。在 Struts 1 中甚至封装表单的 `ActionForm` 和 `Action` 类都是单独的，其实这么做也是有道理的，其一是程序逻辑相当清晰，但是多了许多冗余的代码；其二是方便灵活的开发，比如开发人员在变更了 JSP 页面的输入项时，只需要变更 `ActionForm`。在 Struts 2 中实际上是将 `ActionForm` 和 `Action` 的功能都并入到 `Action` 中了，因此 JSP 页面的输入项变了，`Action` 的属性就会有所变化，但这并不意味着 POJO 类的属性就必须得变。可见，这种整合方式处理并不灵活，且 `Action` 承载了过多的任务。

另一种整合策略是可以将 POJO 类和 Action 类区分开来，POJO 只需要考虑对数据库表数据的封装，至于如何操作则在 Action 类部来定义，并且 Action 还封装 JSP 页面的表单输入数据。这种整合的策略思路清晰、逻辑清楚，提倡读者应用。

## 15.7.2 集成的实现

### 【实例 15-6】用 Struts 2+Hibernate 3 集成实现用户登录功能

有用户要登录时，在输入用户名和密码后提交表单，此时需要到数据库的用户表中校验数据，如果用户名和密码输入正确则验证通过。本例的目标是要用 Struts 2 来进行流程处理，用 Hibernate 来做数据检查。

本例中为实现用户登录的功能，设计的程序流程如图 15-13 所示。

用户在 `login.jsp` 页面的表单中输入用户名、密码，再按“提交”按钮，根据表单的 `action` 属性值，将表单数据提交到 `LoginAction` 这一 `action` 组件；在 `LoginAction` 中做用户检验，检验用户名和密码是否正确就通过 Hibernate 来对数据库中的数据进行操作。如果检验通过，则跳转到 `loginResult.jsp`，显示登录的用户名并报告登录成功；如果检验失败就回转到 `login.jsp`，显示用户名并报告登录失败。

要用 Hibernate 操作数据，就需要设计一个对应于数据库中用户表的 POJO 类，再配以相应的 `hbm.xml` 配置文件，并在 `hibernate.cfg.xml` 做出映射配置，此后就可以在 Action 中通过



Hibernate 的 Session 来快速而简单地操作数据了。

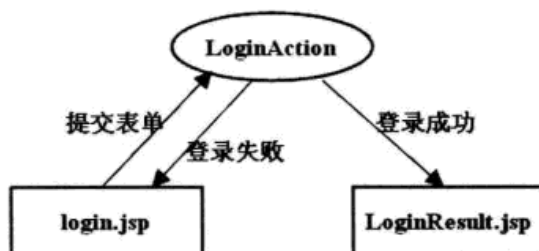


图 15-13 用户登录的程序流程

#### login.jsp

```

<%@ page contentType="text/html;charset=GBK" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head><title>登录系统</title></head>
<body><br><br>
<div align="center">
${message}
<s:form action="Login" method="POST" theme="xhtml">
  <s:textfield name="username" label="用户名"/>
  <s:password name="password" size="21" label="密码"/>
  <s:submit value="提 交"/>
</s:form>
</div>
</body>
</html>

```

message 表示提示消息，用户登录失败后会返回到 login.jsp 页面，message 就用于提醒失败的原因是什么。在后续的代码中读者将会看到，message 实际上是 Action 的一个属性。这个页面的表单 action 属性值为 Login，表示数据将提交到 Login.action，至于 Action 类是谁则需要到 Struts2 的配置文件 struts.xml 中去定位。表单中有一个文本框，用于输入用户名；一个密码框，用于输入用户名对应的密码；一个提交按钮。因为 Struts2 中有默认的页面风格，会自动生成 HTML 代码，因此此处不必再编写表格<table>、<tr>、<td>这样的代码。

接下来用与制作 login.jsp 页面同样的方法再制作一个 JSP 页面 loginResult.jsp，源代码如下。

#### loginResult.jsp

```

<%@ page contentType="text/html;charset=GBK" %>
<html>
<head>
<title>验证通过</title>
</head>
<body>
${message}
</body>
</html>

```

页面成功后的提示信息由 message 表示并输出，实际上 message 是 Action 的一个属性。

本例需要编写三个类：LoginAction 类，这是登录操作的 Action 类；HibernateUtil 类用于生

成 Hibernate 数据库操作的 Session 对象; SysUser 类是与数据库中的用户表对应起来的 POJO 类。HibernateUtil 类在此前已有类此处就不再列出了。

## LoginAction.java

```
package com.csai.action;
import java.util.ArrayList;
import org.hibernate.Query;
import org.hibernate.Session;
import com.csai.db.HibernateUtil;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String username;//用户名
    public String password;//密码
    public String message;//返回 JSP 页面的消息
    @Override
    public String execute() throws Exception {
        //得到 Hibernate 会话对象
        Session sessionHiberante=HibernateUtil.getSession();
        //开始事务
        sessionHiberante.beginTransaction();
        //创建 HQL 查询
        Query hqlQuery=
            sessionHiberante.createQuery("from SysUser where username=?"+
                "and password=?");
        //设置查询语句中的参数值
        hqlQuery.setString(0,username);
        hqlQuery.setString(1,password);
        //得到查询结果
        ArrayList result =(ArrayList)hqlQuery.list();
        //结束事务
        sessionHiberante.getTransaction().commit();
        //判断验证是否通过
        if(result.size()>0){
            message=new String("用户名和密码输入正确,通过验证。");
            return SUCCESS;
        }else{
            message=new String("用户和密码输入有误!");
            return INPUT;
        }
    }
    /**
     *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}
```

LoginAction 类继承自 ActionSupport 类。尽管 Struts 2 声称能较好地支持程序的解耦,也就是说 Action 类可以不继承自任何 Struts 2 框架中的类,可直接在其中编写处理逻辑代码,但是继承 ActionSupport 类会给程序的编写带来许多的方便,比如带来清晰的程序流程、简便的程序开发过程等。



## SysUser.java

```
package com.csai.db;
public class SysUser {
    public long userid;//用户 ID 号
    public String username;//用户名
    public String password;//密码
    /**
     * 此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}
```

相应的数据库建表语句如下:

```
CREATE TABLE [sysuser] (
    [sysuserid] [bigint] IDENTITY (1, 1) NOT NULL ,
    [sysusername] [varchar] (40) COLLATE Chinese_PRC_CI_AS NULL ,
    [sysuserpassword] [varchar] (40) COLLATE Chinese_PRC_CI_AS NULL ,
    CONSTRAINT [PK_sysuser] PRIMARY KEY CLUSTERED
    (
        [sysuserid]
    ) ON [PRIMARY]
) ON [PRIMARY]
```

既然是 Struts 和 Hibernate 集成的案例,而这两种开源软件都需要用到一些配置文件,因此需要编写几个配置文件。Struts 需要有 struts.xml、struts.properties,Web 应用要有 web.xml, Hibernate 要有 hibernate.cfg.xml, SysUser 类这个 POJO 类对应的配置文件有 SysUser.hbm.xml(配置 Java 类与数据库表之间的映射规则)。

### 1. 配置 web.xml 文件

修改“WebContent/WEB-INF”目录下的 web.xml 文件,内容如下:

## web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation=
"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
    <display-name>ch15</display-name>
    <!-- Struts2 过滤器 -->
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <!-- 欢迎页面 -->
    <welcome-file-list>
```

```
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

配置文件中配置了一个名为“struts2”的过滤器，过滤器类为 `org.apache.struts2.dispatcher.FilterDispatcher`，并针对所有的 Web 访问来过滤，因为 `url-pattern` 设置的值为“/\*”。

## 2. 配置 struts.xml 文件

struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="ch15" namespace="" extends="struts-default">
        <action name="Login" class="com.csai.action.LoginAction">
            <result>/loginResult.jsp</result>
            <result name="input">/login.jsp</result>
        </action>
    </package>
</struts>
```

表示包名为 `ch15`，扩展了 `struts-default`，这样就可以充分利用“struts2”已有的特性。`<package>`中配置了一个 Action，名称为 `Login`，Action 类为 `LoginAction`，指出类名时需要给出 Action 类的全路径。

如果 Action 的 `execute()` 方法返回 `success`，在登录的业务中表示用户名密码正确，则页面跳转到 `loginResult.jsp`，`<result>` 标签如果没有给出 `name` 属性值，即表示是默认的 `name="success"`。如果数据校验失败则会返回到 `login.jsp` 页面。成功运行的结果界面如图 15-14 所示。



图 15-14 用户登录界面

## 15.8 小结

Hibernate 架起了 Java 对象与数据库中的关系型数据的桥梁，它可以将关系型数据映射成



Java 对象，将 Java 对象映射成关系型数据，还可以用来做持久化和反持久化。

持久化对象无非也就是简单的 POJO 对象，一个与数据库表对应的包含有若干属性以及属性对应的 `getXxxx()` 方法和 `setXxxx()` 方法的类的实例。持久化对象具有瞬时状态、脱管状态、持久状态三种状态，互相之间可以转换。POJO 对象与数据库表的对应关系通过映射配置文件来定义，即 `.hbm.xml` 文件。

关联关系有一对多、多对一、多对多三种；HQL 是 Hibernate 专用的面向对象的查询语言，语法非常类似于 SQL。

Struts 实现了 MVC 模式，并且对客户端提交的数据校验、程序的执行流程、页面数据的表示方法都进行了封装，以往编程的相当一部分工作通过编写配置文件就可以完成；Hibernate 就像是一个数据库操作的专家，其核心部件实现了 O-R 映射，让 Java 程序员使用数据库就像使用 Java 对象那么方便，Java 对象与数据库表的映射关系也是通过配置文件来完成，数据的持久化工作由 Hibernate 来代劳。

## 15.9 练习

1. 设计一个数据库表，然后在 Web 应用中设计 JSP 页面，利用 Hibernate 结合 JSP 页面实现对该表的增加、删除和更新操作。
2. 在前一题的基础上再设计一个表，两个表之间设计一对多的关系，利用 Hibernate 建立起一对多的双向关联关系，并编写程序调试。

# 16

## 基于 Struts 2+Hibernate 4 实现报到管理系统

---

本章中将采用 Struts 2+Hibernate 4 技术来继续改进第 14 章中实现的报到管理系统。使用 Hibernate 4 技术后，读者会发现开发工作变得更轻松了，Struts 2 框架中的 Action 代码会大幅度减少，不过编写配置文件的工作量会增加一些，但这也给系统带来了良好的灵活性，何况还有 Hibernate 4 这样的数据库数据操作专家来帮忙处理数据操作的问题，开发人员可以进一步得到解放，得到优美的系统架构。

### 16.1 系统设计思想

在第 14 章中用 Struts 2 实现了报到管理系统，相对用纯 JSP 技术实现的系统有了较大的改进，但也存在一些问题：

(1) Action 类的 `execute()` 方法中代码冗长。其实仔细一看其中的代码，大多数的也就是在构造 SQL 语句，生成 SQL 语句对象，然后执行 SQL 语句，总是重复的执行这些工作，能否再简化一些？

(2) 数据表之间有关联关系时，需要做表与表之间的联合查询，或做多次查询才能得到结果记录集，SQL 语句相当长，程序员在编写程序时稍一不慎就写错了，同时调试程序也比较麻烦，特别是在 JSP 页面中输入多个查询条件，要做组合查询时，由于关联到多个表，构造的 `where` 子句也比较长。

(3) 依靠 `DBConn` 类来生成一个数据库连接，在每个 Action 类的 `execute()` 方法中都要打开和关闭连接，而且连接参数就直接写在程序代码中，如果连接参数有所变化，就是重新编译 `DBConn` 类，能否将数据库的连接参数写在配置文件中呢？这样将会具有更好的灵活性。

以上问题都可以通过 Hibernate 来解决，此外，Hibernate 还为异种数据库之间的系统移植



提供了可能,因为操作数据库中数据的 SQL 语句,不是程序员在 Java 代码中写的,程序员写的是面向 Java 对象的 HSQL 语句(当然也可以使用 SQL 语句),再由 Hibernate 根据数据库的方言翻译成 SQL 语句,再做进一步处理,这样可以屏蔽异种数据库之间的差异。

使用 Struts+Hibernate 后的系统架构如图 16-1 所示。读者对比第 14 章中的图 14-1 可知,其实就是将数据操作层所做的事情交给了 Hibernate,而 Hibernate 又通过 POJO 对象与数据库的数据表对应,对应关系则由 hibernate.cfg.xml 及 POJO 类的配置文件来给出。

**提示** 在本章中使用的版本是 Struts 2 和 Hibernate 4,版本不同会稍有差异,不过 Struts 1 和 Struts 2 差异是很大的,但在小的版本之间差异非常小,比如 Struts 2.0.11 和 Struts 2.1.2。

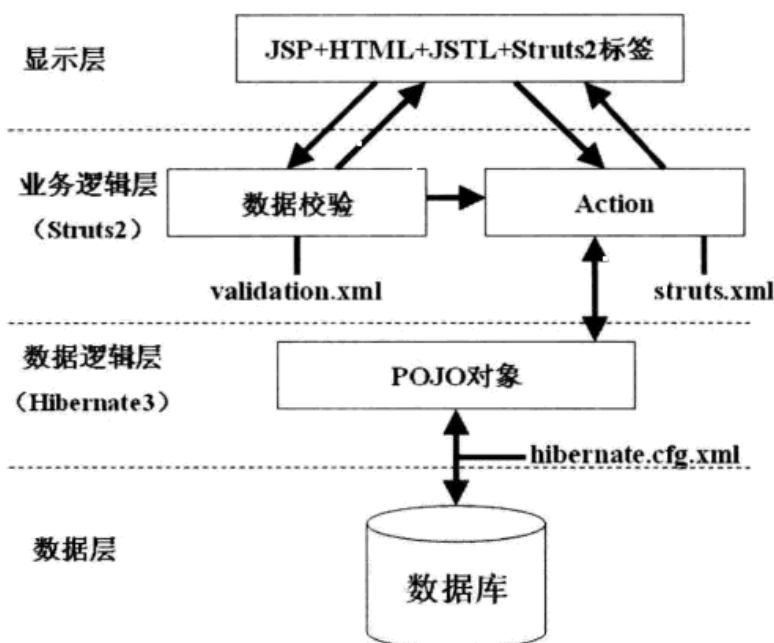


图 16-1 基于 Struts+Hibernate 的系统架构

## 16.2 系统开发框架搭建

### 16.2.1 在 Eclipse 中搭建 Web 应用的开发框架

在搭建 Web 应用的开发框架前,请读者确保机器已安装并配置好 JDK、Eclipse、Tomcat,准备好 Struts 和 Hibernate 开发包,在开发过程中需要使用到这些开发工具与软件包。

当前 Web 应用要使用到 Struts 和 Hibernate 框架,因此请将要用到的 Struts 开发包及 Hibernate 开发包复制到当前 Web 应用的“WEB-INF\lib”目录中。

本章中的工程要用到的 jar 包与第 15 章中的相同,因此可将第 15 章中工程的“WEB-INF\lib”目录中的 jar 复制过来即可。

本例中,当前工程的目录为“d:\eclipse\workspace\ch16”,则默认当前 Web 应用目录为

“d:\eclipse\workspace\ch16\WebContent”，可将 jar 文件放置于“d:\eclipse\workspace\ch16\WebContent\WEB-INF\lib”目录中。读者请根据自己配置的情况对应操作。



**提示** 如果实在不清楚要放哪些 jar 文件，可以将所有 jar 文件都复制过去。

在 Eclipse 的树形菜单中选中工程名“ch16”，按 F5 键（或使用右键快捷菜单）刷新，搭建完成的工程 Web 应用“WEB-INF/lib”目录下的情况如图 16-2 所示。



图 16-2 Web 应用“WEB-INF/lib”目录下的情况



**提示** 复制完 jar 包以后，注意要刷新后才能在 Eclipse 的树形菜单中看到更新的最新情况。刷新的方法是选中“ch16”节点，按 F5 键，或使用右键快捷菜单中的“刷新”菜单。

## 16.2.2 准备 Hibernate 相关的配置文件与包

本系统包的设置和第 14 章中的工程操作一致，仍设置三个包：com.csai.action、com.csai.db、com.csai.POJO。com.csai.action 包中存放所有的 Action 类，com.csai.db 存放与数据库操作相关的类，com.csai.POJO 存放所有的 POJO 类。

在当前工程中分别建立 com.csai.action、com.csai.POJO、com.csai.action 这三个包，如果有



的话可保持不变, 如果没有就新建这些包。建包的方法可参见图 14-3。

有关 Struts 的配置文件保持不变, 因为根据本章系统的系统设计思想来看, 业务逻辑层不需要改变, 仅改变数据逻辑层的内容, 通过 Action 类来与 Hibernate 协同工作。

在“src”目录下新建一个 hibernate.cfg.xml 文件, 这是与 Hibernate 相关的配置文件, 在其中可配置数据库的连接参数, 以及与 POJO 类对应的配置文件。新建 hibernate.cfg.xml 文件的步骤可参见图 14-5。hibernate.cfg.xml 的内容如下所示。

hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <!--会话工厂配置-->
    <session-factory>
        <!--数据库方言-->
        <property name="dialect">
            org.hibernate.dialect.SQLServerDialect
        </property>
        <!--数据库登录用户名-->
        <property name="connection.username">sa</property>
        <!--数据库用户登录密码-->
        <property name="connection.password">123</property>
        <!--数据库连接字符串-->
        <property name="connection.url">
            jdbc:sqlserver://localhost:1433;DatabaseName=RegisterSystem
        </property>
        <!--是否显示 SQL 语句-->
        <property name="show_sql">true</property>
        <!--POJO 对象对应的映射文件-->
        <mapping resource="com/csai/POJO/AdminUser.hbm.xml" />
        <mapping resource="com/csai/POJO/Student.hbm.xml" />
        <mapping resource="com/csai/POJO/Bedchamber.hbm.xml" />
        <mapping resource="com/csai/POJO/ClassTa.hbm.xml" />
        <mapping resource="com/csai/POJO/Speciality.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

<session-factory>标签中的内容为会话工厂配置, Hibernate 中的会话就好比是 JDBC 中的 Connection, 也就是数据会话, 在一个 Web 应用中可共享使用一个会话工厂来生成会话。连接的是何种数据库通过 dialect 属性来配置; 数据库驱动程序通过 connection.driver\_class 属性来配置; 数据库登录用户名通过 connection.username 来配置; 数据库用户登录密码通过 connection.password 属性来配置; 数据库连接字符串通过 connection.url 属性来配置。读者可根据自己本地设置的情况来修改这些参数。

<mapping>标签配置 POJO 类对应的映射文件, 这里相关的数据库表有五个, 也就对应着有五个 POJO 类对象, 因此就有五个配置文件, 文件的名称均为“POJO 类名.hbm.xml”, 均与 POJO 类处于同一目录下。

### 16.2.3 设计 POJO 类与映射配置文件

POJO 类与数据库表是一一对应地，但属性与数据库表字段并不完全相同。名称允许不同，而通过映射文件来配置 POJO 类属性与数据库表字段之间的对应关系。下面来一一参看。

#### 1. AdminUser 类

AdminUser 类表示系统用户，与数据库中的 adminuser 表对应，它的情况最简单，因为没有与其他的表发生关联关系。AdminUser 类的代码如下所示。

AdminUser.java

```
package com.csai.POJO;
public class AdminUser {
    public String adminUserName;
    public String adminUserPassword;
    public int adminUserRole;
    /**
     * 此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}
```

在编写代码时，属性对应的 setXxxx() 方法和 getXxxx() 方法没有必要手写，可以由 Eclipse 生成，为形成良好的编程习惯以及免除一些不必要的麻烦。

在 Eclipse 中自动生成属性对应的 setXxxx() 方法和 getXxxx() 方法的操作步骤是：在已打开的源代码编辑框中单击右键，在弹出的快捷菜单中选中“Source”→“Generator Getters and Setters...”，在弹出的对话框中选择要生成的方法即可。

AdminUser 类对应的映射文件内容如下所示。

AdminUser.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
<class name="com.csai.POJO.AdminUser" table="adminuser" schema="dbo">
    <id name="adminUserName" type="text"
        column="adminusername"/>
    <property name="adminUserPassword" type="text"
        column="adminuserpassword" />
    <property name="adminUserRole" type="integer"
        column="adminuserrole"/>
</class>
</hibernate-mapping>
```

com.csai.POJO.AdminUser 类与数据库 dbo 模式中的 adminuser 表对应起来；主键为 adminUserName 属性，对应着 adminuser 表的 adminusername 字段，数据类型为文本 text；属性 adminUserPassword 对应着 adminuser 表的 adminuserpassword 字段，数据类型为文本 text；



属性 `adminUserRole` 对应着 `adminuser` 表的 `adminuserrole` 字段，数据类型为整型 `integer`。

## 2. Bedchamber 类

这个类表示宿舍，与数据库中的 `Bedchamber` 表相对应，类的源代码如下所示。

Bedchamber.java

```
package com.csai.POJO;
import java.util.List;
public class Bedchamber {
    public Integer bedchamberId;
    public String bedchamberName;
    public List<Student> studentList;
    /**
     * 此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}
```

`Bedchamber` 类对应的映射配置文件内容如下所示。

Bedchamber.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
<class name="com.csai.POJO.Bedchamber" table="Bedchamber" schema="dbo">
    <id name="bedchamberId" type="integer" column="BedchamberId">
        <generator class="identity"/>
    </id>
    <property name="bedchamberName" type="text" column="BedchamberName"/>
    <list name="studentList" inverse="true" cascade="all">
        <key column="BedchamberId"/>
        <index column="StudentId"/>
        <one-to-many class="com.csai.POJO.Student"/>
    </list>
</class>
</hibernate-mapping>
```

`com.csai.POJO.Bedchamber` 类对应着数据库 `dbo` 模式的 `Bedchamber` 表。属性 `bedchamberId` 与 `Bedchamber` 表的字段 `BedchamberId` 对应，为关键字，数据类型为 `integer`，`identity` 表示该值由数据库自动生成。`bedchamberName` 属性与 `Bedchamber` 表的字段 `BedchamberName` 对应，数据类型为 `text`。

属性 `studentList` 表示该宿舍中的学生列表，关系由相关联的类来控制，并且叠加修改。数据库表中的关键字段为 `BedchamberId`。由于是 `List` 列表，因此需要用 `<index>` 标签指出索引，以根据此索引来排列顺序，此处设为 `StudentId` 字段，表示根据学生的 ID 号来排序。`<one-to-many>` 表示一对多的关系，即一个宿舍中可以有多名学生，对应的学生类为 `com.csai.POJO.Student`。

属性 `studentList` 使用了泛型，这样 `Hibernate` 就能明确地知道 `studentList` 列表元素的数据类型为 `Student`。

### 3. ClassTa 类

ClassTa 类表示班级，与数据库中的 ClassTa 表相对应，类的源代码如下所示。

ClassTa.java

```
package com.csai.POJO;
import java.util.List;
public class ClassTa {
    public Integer classId;
    public String className;
    public List<Student> studentList;
    /**
     *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}
```

ClassTa 类相应的映射配置文件内容如下所示。

ClassTa.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
<class name="com.csai.POJO.ClassTa" table="ClassTa" schema="dbo">
    <id name="classId" type="integer" column="ClassId">
        <generator class="identity"/>
    </id>
    <property name="className" type="text" column="ClassName"/>
    <list name="studentList" inverse="false" cascade="all" table="Student">
        <key column="ClassId"/>
        <index column="StudentId"/>
        <one-to-many class="com.csai.POJO.Student"/>
    </list>
</class>
</hibernate-mapping>
```

com.csai.POJO.ClassTa 类对应着数据库 dbo 模式的 ClassTa 表。属性 classId 与 ClassTa 表的字段 ClassId 对应，为关键字，数据类型为 integer，identity 表示该值由数据库自动生成。className 属性与 ClassTa 表的字段 ClassName 对应，数据类型为 text。

属性 studentList 表示该班中的学生列表，关系由相关联的类来控制，并且叠加修改，关联的表为 Student。数据库表中的关键字段为 ClassId。由于是 List 列表，因此需要用<index>标签指出索引，以根据此索引来排列顺序，此处设为 StudentId 字段，表示根据学生的 ID 号来排序。<one-to-many>表示一对多的关系，即一个班中可以有多名学生，对应的学生类为 com.csai.POJO.Student。

### 4. Speciality 类

Speciality 类表示专业，与数据库中的 Speciality 表相对应，类的源代码如下所示。



## Speciality.java

```
package com.csai.POJO;
import java.util.List;
public class Speciality {
    public Integer specialityId;
    public String specialityName;
    public List<Student> studentList;
    /**
     *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}
```

Speciality 类相应的映射配置文件内容如下所示。

## Speciality.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
<class name="com.csai.POJO.Speciality" table="Speciality" schema="dbo">
    <id name="specialityId" type="integer" column="SpecialityId">
        <generator class="identity"/>
    </id>
    <property name="specialityName" type="text" column="SpecialityName"/>
    <list name="studentList" inverse="true" cascade="all">
        <key column="SpecialityId"/>
        <index column="StudentId"/>
        <one-to-many class="com.csai.POJO.Student"/>
    </list>
</class>
</hibernate-mapping>
```

com.csai.POJO.Speciality 类对应着数据库 dbo 模式的 Speciality 表。属性 specialityId 与 Speciality 表的字段 SpecialityId 对应，为关键字，数据类型为 integer，identity 表示该值由数据库自动生成。specialityName 属性与 Speciality 表的字段 specialityName 对应，数据类型为 text。

属性 studentList 表示该专业中的学生列表，关系由相关联的类来控制，并且叠加修改。数据库表中的关键字段为 SpecialityId。由于是 List 列表，因此需要用<index>标签指出索引，以根据此索引来排列顺序，此处设为 StudentId 字段，表示根据学生的 ID 号来排序。<one-to-many>表示一对多的关系，即一个专业中可以有多名学生，对应的学生类为 com.csai.POJO.Student。

## 5. Student 类

Student 类表示学生，与数据库中的 Student 表相对应，类的源代码如下所示。

## Student.java

```
package com.csai.POJO;
import java.sql.Date;
public class Student {
    Long studentId;
```



```

String studentName;
Speciality speciality;
Bedchamber bedchamber;
ClassTa classTa;
String matriNo;
Float payAmount;
Integer payOK;
Date registDate;
/**
 *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
 */
}

```

Student 类对应的映射配置文件内容如下所示。

#### Student.hbm.xml

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>
<class name="com.csai.POJO.Student" table="Student" schema="dbo">
    <id name="studentId" type="long" column="StudentId">
        <generator class="identity"/>
    </id>
    <property name="studentName" type="text" column="StudentName" not-null="true"/>
    <property name="matriNo" type="text" column="MatriNo"/>
    <property name="payAmount" type="float" column="PayAmount" />
    <property name="payOK" type="integer" column="PayOK"/>
    <property name="registDate" type="date" column="RegistDate"/>
    <many-to-one name="classTa" column="classId"
class="com.csai.POJO.ClassTa" outer-join="false"/>
    <many-to-one name="speciality"
column="SpecialityId" class="com.csai.POJO.Speciality"/>
    <many-to-one name="bedchamber" column="BedchamberId"
class="com.csai.POJO.Bedchamber"/>
</class>
</hibernate-mapping>

```

com.csai.POJO.Student 类对应着数据库 dbo 模式的 Student 表。属性 studentId 与 Student 表的字段 StudentId 对应，为关键字，数据类型为 long，identity 表示该值由数据库自动生成。studentName 属性与 Student 表的字段 StudentName 对应，数据类型为 text，not-null="true" 表示内容不能为空。

matriNo 属性与 Student 表的字段 MatriNo 对应，数据类型为 text。payAmount 属性与 Student 表的字段 PayAmount 对应，数据类型为 float。payOK 属性与 Student 表的字段 PayOK 对应，数据类型为 integer。registDate 属性与 Student 表的字段 RegistDate 对应，数据类型为 date。

Student 与 ClassTa、Speciality、Bedchamber 存在着多对一的关系，也就是说多个学生在同一个班，多个学生在同一个专业，多个学生在同一个宿舍。

<many-to-one>标签的 many 属性指出了 POJO 类中的属性名，column 属性指出相关联的数



数据库表的关键字段，class 属性指出相关联的类全路径。

**提示** 编写映射配置文件时，要特别注意的就是哪个属性的值是 POJO 类的，哪个属性的值是数据库表的，如果实在混淆不清的话，不如在设计时就将数据库的字段名和 POJO 类的属性名保持一致。不过，这在实际工程中往往很难做到，因为数据库设计人员和 Java 程序开发人员并不一定会是同一个人，各人的编程习惯可能有所不同，比如名称命名的问题。因此，建议读者还是详细掌握配置文件的各个常用属性代表什么含义。

### 16.2.4 开发会话工厂类

在 com.csai.db 包中新建一个类 HibernateUtil，源代码如下所示。

HibernateUtil.java

```
package com.csai.db;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    public static final ThreadLocal session = new ThreadLocal();
    static {
        try {
            // 根据配置文件 hibernate.cfg.xml 创建 SessionFactory
            sessionFactory = new Configuration().configure().buildSessionFactory();
            System.out.println("初始化 SessionFactory 成功。");
        } catch (Throwable ex) {
            System.err.println("初始化 SessionFactory 失败。" + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static Session getSession() {
        Session s = (Session) session.get();
        // 获得一个新的 Session
        if (s == null) {
            s = sessionFactory.openSession();
            session.set(s);
        }
        return s;
    }
}
```

在此类的静态代码中，Configuration().configure().buildSessionFactory() 会自动加载 hibernate.cfg.xml 来生成一个会话工厂，这段代码为什么要设计成静态的呢？这样的话这段代码就会只在第一次生成该类的实例时执行，此后就不再执行了。getSession()这个方法用于使用会话工厂来生成一个会话。



## 16.3 系统各功能点的实现

从系统的设计思想来看，显示层并没有做任何改变，因此本章中实现的系统相对第 12 章实现的系统而言，很少需要修改 JSP 网页中的代码，修改较大的是 Action 类的代码。

### 16.3.1 用户登录功能的实现

用户登录功能要修改的就是 LoginAction 类的代码了，特别是 execute() 方法中的代码。

LoginAction.java

```
package com.csai.action;
import java.util.ArrayList;
import org.hibernate.Query;
import org.hibernate.Session;
import com.csai.POJO.AdminUser;
import com.csai.db.HibernateUtil;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String adminusername; //用户名
    public String adminuserpassword; //密码
    public String action; //动作
    public String errorMsg; //报错信息
    @SuppressWarnings("unchecked")
    @Override
    public String execute() {
        if("login".equals(action)){
            try{
                //构造 HSQL 语句并查询
                String hsql="from AdminUser where adminusername=? and "+
                    "adminuserpassword=?";
                Session sessionHiberante=HibernateUtil.getSession();
                sessionHiberante.beginTransaction();
                Query query=sessionHiberante.createQuery(hsql);
                query.setString(0,adminusername);
                query.setString(1,adminuserpassword);
                ArrayList<AdminUser> result =(ArrayList<AdminUser>)query.list();
                sessionHiberante.getTransaction().commit();
                if(result.size()>=1){//如果用户名和密码正确
                    ActionContext.getContext().getSession().put(
                        "adminusername",result.get(0).getAdminUserName());
                    ActionContext.getContext().getSession().put(
                        "adminuserpassword",result.get(0).getAdminUserPassword());
                    ActionContext.getContext().getSession().put(
                        "adminuserrole",result.get(0).getAdminUserRole());
                    return SUCCESS;
                }else{
```



```

        errmsg=new String("用户名或密码输入有误");
    }
    }catch(Exception e){
        errmsg=new String("数据库操作有误");
        e.printStackTrace();
    }
    }
    return INPUT;
}
/**
 *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
 */
}

```

HSQL 语句比 SQL 语句相对更加简洁。比如此处的 HSQL 代码为：

```
from AdminUser where adminUserName=? and adminUserPassword=?
```

此处，AdminUser 是指 POJO 类名，adminUserName 和 adminUserPassword 是 POJO 类的属性，也就是说编写 HSQL 语句时，就感觉是针对 POJO 对象的操作，而不是针对数据库表。

HSQL 语句对象也可以有参数，设置参数时使用了如下的语句：

```
query.setString(0,adminusername);
query.setString(1,adminuserpassword);
```

用户名 adminusername 和密码 adminuserpassword 是从用户登录表单提交过来的，它们是 Action 的属性，Struts 会根据 Action 属性的名称与提交的表单项名称去配备，从而设定 Action 属性的值。



**提示** HSQL 语句对象中的参数编号从 0 开始，这和 SQL 语句对象不同，SQL 语句对象是从 1 开始的。

查询出数据后得到的是一个数组列表，如果数组列表的长度大于 1 则表示用户名和密码正确，接着在 session 中保存用户名、密码和角色，记录的语句示例如下：

```
ActionContext.getContext().getSession().put(
    "adminusername",result.get(0).getAdminUserName());
```

result.get(0).getAdminUserName()表示得到 result 列表的第一个元素的用户名。需要注意的是，列表的索引号是从 0 开始的。

### 16.3.2 专业基础数据管理功能的实现

专业基础数据管理功能要实现对专业表（Speciality）的增加、删除操作，来看 Action 类代码的变化。

SpecialityAction.java

```
package com.csai.action;
import java.util.ArrayList;
import java.util.Map;
```



```

import org.hibernate.Query;
import org.hibernate.Session;
import com.csai.POJO.Speciality;
import com.csai.db.HibernateUtil;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class SpecialityAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String specialityname;//专业名称
    public int specialityid;//专业 ID 号
    public String action;//动作

    @Override
    public String execute() throws Exception {
        Session sessionHibernate=HibernateUtil.getSession();
        sessionHibernate.beginTransaction();
        Map request = (Map)ActionContext.getContext().get("request");
        //----如果是增加一个专业---
        if("add".equals(action)){
            String hsql="from Speciality where specialityname=?";
            Query query=sessionHibernate.createQuery(hsql);
            query.setString(0,specialityname);
            ArrayList<Speciality> specArray =(ArrayList<Speciality>)query.list();
            if(specArray.size()<=0){//没有这个专业
                Speciality spec=new Speciality();
                spec.setSpecialityName(specialityname);
                sessionHibernate.save(spec);
            }
        }
        //----如果是删除一个专业----
        if("del".equals(action)){
            String hsql="from Speciality where specialityid=?";
            Query query=sessionHibernate.createQuery(hsql);
            query.setInteger(0,specialityid);
            ArrayList<Speciality> specArray =(ArrayList<Speciality>)query.list();
            if(specArray.size()>=1)
                sessionHibernate.delete(specArray.get(0));
        }
        //----查询出已有的专业数据----
        String hsql="from Speciality";
        Query query=sessionHibernate.createQuery(hsql);
        ArrayList<Speciality> specialityArray=(ArrayList<Speciality>)query.list();
        request.put("specialityArray", specialityArray);
        sessionHibernate.getTransaction().commit();
        return SUCCESS;
    }
    /**
     *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}

```



使用 Hibernate 后,可以发现以上 Action 类的代码变得简洁多了。比如在增加一个专业时,先用如下的 HSQL 语句:

```
from Speciality where specialityname=?
```

使用以上 HSQL 语句进行查询,如果结果列表的长度小于或等于 0,表示暂时还没有这个专业,再使用如下的语句:

```
Speciality spec=new Speciality();
spec.setSpecialityName(specialityname);
sessionHibernate.save(spec);
```

第一句新建了一个 Speciality 对象 spec;第二句设置 spec 对象的专业名称属性 specialityName 的值;第三句利用 Hibernate 会话对象来保存 spec 对象,这样 Hibernate 就会自动生成 insert 语句往 Speciality 表中插入一条记录。

在删除一个专业时,先使用如下的 SQL 语句:

```
from Speciality where specialityid=?
```

通过 Hibernate 查询得到要删除的 Speciality 对象,再使用如下的语句来删除这个对象:

```
sessionHibernate.delete(specArray.get(0));
```

specArray 是一个 Speciality 对象列表, get(0)则表示得到第一个元素, Hibernate 会自动作持久化处理,从而从 Speciality 表中删除记录。

### 16.3.3 录取学生名册基础数据管理功能的实现

这个功能的实现相对要复杂一些,因为查询数据时是组合查询,并且还要做数据分页处理。MatriAction 类的代码相当冗长,程序员稍有不慎就会写错,要求有比较高的专业水准,而且调试程序也比较麻烦。

MatriAction.java

```
package com.csai.action;
import java.util.ArrayList;
import java.util.Map;
import org.hibernate.Query;
import org.hibernate.Session;
import com.csai.POJO.Speciality;
import com.csai.POJO.Student;
import com.csai.db.HibernateUtil;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class MatriAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String action;//动作
    public Integer specialityid;//专业 ID 号
    public String matrino;//录取通知书号
    public String studentname;//学生姓名
    public Integer currentpage=1;//当前页码
    public Long studentid;//学生 ID 号
```



```

@Override
public String execute() throws Exception {
    if(studentname!=null&&studentname.length()!=0)
        studentname=studentname.trim();
    if(action!=null&&action.length()!=0)
        action=action.trim();
    if(matrino!=null&&matrino.length()!=0)
        matrino=matrino.trim();
    Session sessionHibernate=HibernateUtil.getSession();
    sessionHibernate.beginTransaction();
    Map request = (Map)ActionContext.getContext().get("request");
    //----如果是增加一个学生---
    if("add".equals(action)){
        String hsql="from Student where matriNo=? and studentName=?"+
            " and speciality.specialityId=?";
        Query query=sessionHibernate.createQuery(hsql);
        query.setString(0,matrino);
        query.setString(1,studentname);
        query.setInteger(2,specialityid);
        ArrayList<Student> stuArray =(ArrayList<Student>)query.list();
        if(stuArray.size()<=0){//没有此学生
            Student stu=new Student();
            stu.setMatriNo(matrino);
            stu.setStudentName(studentname);
            hsql="from Speciality where specialityId="+specialityid;
            Query query1=sessionHibernate.createQuery(hsql);
            Speciality spec=((ArrayList<Speciality>)query1.list()).get(0);
            stu.setSpeciality(spec);
            sessionHibernate.save(stu);
        }
    }
    //----如果是删除一个学生---
    if("del".equals(action)){
        String hsql="from Student where studentId="+studentid;
        Query query=sessionHibernate.createQuery(hsql);
        ArrayList<Student> stuArray =(ArrayList<Student>)query.list();
        if(stuArray.size()==1)
            sessionHibernate.delete(stuArray.get(0));
    }
    //----如果是查询数据----
    int pagesize=5;//每页记录条数
    int pagecount=0;//总页数
    int recount=0;//总记录条数
    if("select".equals(action)){
        String hsql=null;
        String hsqlwhere=" where 1=1 ";//where 子句
        if(studentname!=null&&studentname.length()>=1)
            hsqlwhere+=" and s.studentName like '%"+studentname+"%'";
        if(specialityid!=null&&specialityid.intValue()!=0)
            hsqlwhere+=" and s.speciality.specialityId="+specialityid;
        String hsqlcount="select count(*) from Student s "+hsqlwhere;
        recount = ((Long)sessionHibernate.createQuery(
            hsqlcount).uniqueResult()).intValue();
    }
}

```



```

        //----得到总页数----
        if(recount%pagesize==0)//能整除
            pagecount=recount/pagesize;
        else//不能整除
            pagecount=(int)(recount/pagesize)+1;
        hsql="from Student s "+hsqlwhere+" order by s.studentId desc";
        Query query=sessionHibernate.createQuery(hsql);
        query.setFirstResult((currentpage-1)*pagesize);
        query.setMaxResults(pagesize);
        ArrayList<Student> stuArray =(ArrayList<Student>)query.list();
        request.put("stuArray", stuArray);
    }
    //----查询出专业数据----
    String hsql="from Speciality";
    Query query=sessionHibernate.createQuery(hsql);
    ArrayList<Speciality> specialityArray =(ArrayList<Speciality>)query.list();
    request.put("specialityArray", specialityArray);
    //----将数据放入 request----
    request.put("pagesize",pagesize);
    request.put("pagecount",pagecount);
    request.put("currentpage",currentpage);
    request.put("recount",recount);
    sessionHibernate.getTransaction().commit();
    return SUCCESS;
}
/**
 * 此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
 */
}

```

可见代码相对简洁多了。当要增加一个学生的录取信息时，首先要判断是否已经有此条录取信息，如果没有就增加。但是麻烦的事情又来了，增加的学生录取信息，数据要写入到 **Student** 表中，然而 **Student** 类中录取专业使用的是对象型属性，那怎么办呢？来看是如何一步一步进行操作的。

下面是先做查询的 HSQL 语句：

```
from Student where matrinNo=? and studentName=? and speciality.specialityId=?
```

查询结果如果没有记录则表示尚没有此条信息，再使用如下语句做添加处理：

```

Student stu=new Student();
stu.setMatriNo(matrino);
stu.setStudentName(studentname);
hsql="from Speciality where specialityId="+specialityid;
Query query1=sessionHibernate.createQuery(hsql);
Speciality spec=((ArrayList<Speciality>)query1.list()).get(0);
stu.setSpeciality(spec);
sessionHibernate.save(stu);

```

先是新建了一个 **Student** 对象 **stu**，接着设置了对象 **stu** 的录取通知书号、学生姓名。再用如下的 HSQL 语句做了一个查询：

```
"from Speciality where specialityId="+specialityid;
```

这样即可查询出该学生被录取的专业对应的 **Speciality** 对象 **spec**，再用如下的语句建立对象 **stu** 和对象 **spec** 之间的关联关系：

```
stu.setSpeciality(spec);
```

至此，两者被关联起来。再使用如下的语句持久化对象：

```
sessionHibernate.save(stu);
```

这样就可以正确地添加一个学生的录取信息了。删除一个学生的过程比较简单，也就是先构造查询 **HSQL** 语句，从而得到要删除的学生对象，再使用 **Hibernate** 会话对象的 **delete()** 方法删除这个学生对象，数据即会从数据库表中删除。

查询数据由于要做分页处理，**HSQL** 语句的 **where** 子句会长一些，这里采用了一种有趣的办法来降低 **HSQL** 语句构造的复杂度。先将 **HSQL** 语句的 **where** 子句设为如下的语句：

```
String hsqlwhere=" where 1=1 ";
```

“**1=1**”这个等式恒成立，因此值为 **true**，但是为什么要这样做呢？这是因为在这个 **where** 子句之后再加其他条件，就只需要考虑增加 **and** 子句，而不必每增加一个 **where** 子句都要先做一下判断，从而决定是否要使用 **and** 保留字。比如：

```
if(studentname!=null&&studentname.length()>=1)
    hsqlwhere+=" and s.studentName like '%" + studentname + "%'";
```

如果 **Action** 属性 **studentname** 的值不为空且长度大于等于 1，则表示 **where** 子句要增加学生姓名这一查询条件，为做模糊查询需要使用 **like** 子句，增加这样的 **and** 条件子句到 **where** 子句中即可。

**Hibernate** 做分页查询非常简便，在查询数据前需要设置查询对象的两个属性值：

```
query.setFirstResult((currentpage-1)*pagesize);
query.setMaxResults(pagesize);
```

**setFirstResult()** 用于设置当前页的第一条记录在查询结果中的第一条记录号；**setMaxResults()** 方法用于设置当前页的最大记录条数。至于如何构造查询出当前数据的 **SQL** 语句，以及如何查询出当前页数据，这些工作就都交给 **Hibernate** 去做吧。

### 16.3.4 其他基础数据管理功能的实现

宿舍基础数据管理、班级基础数据管理、用户管理功能的实现方法和专业基础数据管理功能的实现方法相同，这里就不再重复叙述了，读者可在读懂“12.4.2 专业基础数据管理功能的实现”一节的基础上再查看宿舍基础数据管理、班级基础数据管理功能的源代码就会很容易了。

### 16.3.5 学生报到状况查询功能的实现

学生报到状况查询功能只有一个查询条件，那就是学生姓名。下面来看 **Action** 类有些什么变化。



## RegStatusAction.java

```

package com.csai.action;
import java.util.ArrayList;
import java.util.Map;
import org.hibernate.Query;
import org.hibernate.Session;
import com.csai.POJO.Student;
import com.csai.db.HibernateUtil;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class RegStatusAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String studentname;//学生姓名
    public String action;//动作
    @Override
    public String execute() {
        if("select".equals(action)){
            //----查询出已有的数据----
            try{
                String hsql="from Student ";
                if(studentname!=null&&studentname.length()!=0)
                    hsql+="where studentName like '%" +studentname+"%";
                Session sessionHiberante=HibernateUtil.getSession();
                sessionHiberante.beginTransaction();
                Query query=sessionHiberante.createQuery(hsql);
                ArrayList<Student> stuArray =(ArrayList<Student>)query.list();
                sessionHiberante.getTransaction().commit();
                Map<String,ArrayList<Student>> request =
                    (Map<String,ArrayList<Student>>)
                    ActionContext.getContext().get("request");
                request.put("stuArray", stuArray);
            }catch(Exception e){
                e.printStackTrace();
            }
        }
        return SUCCESS;
    }
    /**
     *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}

```

查询条件仅一个，因此 HSQL 语句也是比较简单的，此处代码就不再详细解释了。

### 16.3.6 报到分班功能的实现

报到分班功能中同样修订最大的就是 ClassAdminAction 类，该类的源代码如下所示。

## ClassAdminAction.java

```

package com.csai.action;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

```



```

import org.hibernate.Query;
import org.hibernate.Session;
import com.csai.POJO.ClassTa;
import com.csai.POJO.Student;
import com.csai.db.HibernateUtil;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class ClassAdminAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String studentname;//学生姓名
    public String action;//动作
    public String matrino;//录取通知书号
    public ArrayList<Student> stuParamArray;//学生列表

    @Override
    public String execute() throws Exception {
        Session sessionHibernate=HibernateUtil.getSession();
        sessionHibernate.beginTransaction();
        //----构造查询的 SQL 语句----
        String hsqlwhere=new String("");
        String hsql=new String("");
        if("select".equals(action)){//如果是查询操作
            if(studentname!=null&&studentname.trim().length()!=0)
                hsqlwhere="where studentName like '%" +studentname.trim()+"%' ";
            if(hsqlwhere!=null&&hsqlwhere.length()!=0){
                if(matrino!=null&&matrino.trim().length()!=0)
                    hsqlwhere+=" and matriNo like '%" +matrino.trim()+"%' ";
            }else{
                if(matrino!=null&&matrino.trim().length()!=0)
                    hsqlwhere=" where matriNo like '%" +matrino.trim()+"%' ";
            }
            hsql="from Student "+hsqlwhere;
            Query query=sessionHibernate.createQuery(hsql);
            List<Student> stuArray =(List<Student>)query.list();
            Map request = (Map)ActionContext.getContext().get("request");
            request.remove("stuArray");
            request.put("stuArray", stuArray);
            //----查询出已有的专业数据----
            hsql="from ClassTa";
            Query queryClass=sessionHibernate.createQuery(hsql);
            List<ClassTa> classArray =(List<ClassTa>)queryClass.list();
            request.remove("classArray");
            request.put("classArray", classArray);
        }
        //----设置分班情况----
        if(stuParamArray!=null&&"update".equals(action)){
            for(int i=0;i<stuParamArray.size();i++){
                if(stuParamArray.get(i).getClassTa()!=null&&
                    stuParamArray.get(i).getClassTa().getClassId()!=0){
                    String hsqlstr="from ClassTa where classid="+
                        stuParamArray.get(i).getClassTa().getClassId();
                    Query query=sessionHibernate.createQuery(hsqlstr);
                    ArrayList<ClassTa> claArray =(ArrayList<ClassTa>)query.list();
                }
            }
        }
    }
}

```



```

        if(claArray.size()>=1){
            hsqlstr="from Student where StudentId="+
            stuParamArray.get(i).getStudentId();
            Query queryStu=sessionHibernate.createQuery(hsqlstr);
            ArrayList<Student> stuArray =
            (ArrayList<Student>)queryStu.list();
            if(stuArray.size()>=1){
                Student stu=stuArray.get(0);
                stu.setClassTa(claArray.get(0));
                sessionHibernate.save(stu);
            }
        }
    }
}
sessionHibernate.getTransaction().commit();
return SUCCESS;
}
/**
 *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
 */
}

```

报到分班功能中，如果要做数据查询，则查询的条件只有两个，一个是姓名，另一个是录取通知书号，再根据这两个条件构造 HSQL 语句并查询，查询结果即为符合条件的学生。

如果要设置分班情况，则根据接收到的数组 `stuParamArray` 的情况来判断是否分班。分班这个动作本身跟简单。

```

stuParamArray.get(i).getClassTa()!=null&&stuParamArray.get(i).getClassTa().get
etClassId()!=0

```

这是 if 语句的判定条件，表示 Action 类中 `stuParamArray` 列表中的第 i 个成员的班级对象不为空，且班级号不等于 0 则表示数据校验通过了。然后再用 `ClassTa` 类来获得指定的 `ClassTa` 对象，此外还要获得学生 `Student` 对象，因为两者是相互关联的。看如下的语句：

```

Student stu=stuArray.get(0);
stu.setClassTa(claArray.get(0));
sessionHibernate.save(stu);

```

第一句话得到 `Student` 对象 `stu`；第二句话设置对象 `stu` 和对象 `claArray.get(0)` 之间的关联关系，也就是设置学生所属的班级；第三句话将 `stu` 对象持久化，`Hibernate` 则会生成 SQL 语句（update 语句）并执行。

### 16.3.7 收费情况登记功能的实现

要进行收费情况登记首先要查询出要登记的学生，这就需要修改 `AcceptMoneyAction` 类中的代码。

AcceptMoneyAction.java

```

package com.csai.action;
import java.util.ArrayList;

```



```

import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.ServletActionContext;
import org.hibernate.Query;
import org.hibernate.Session;
import com.csai.POJO.Student;
import com.csai.db.HibernateUtil;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class AcceptMoneyAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String studentname; // 学生姓名
    public String action; // 动作
    public String matrino; // 录取通知书号
    public ArrayList<Student> stuParamArray;
    @Override
    public String execute() throws Exception {
        Session sessionHibernate = HibernateUtil.getSession();
        sessionHibernate.beginTransaction();
        // ---- 查询数据操作 ----
        String hsqlwhere = new String("");
        String hsql = new String("");
        if("select".equals(action)) { // 如果是查询操作
            if(studentname != null && studentname.trim().length() != 0)
                hsqlwhere = "where StudentName like '%" + studentname.trim() + "%' ";
            if(hsqlwhere != null && hsqlwhere.length() != 0) {
                if(matrino != null && matrino.trim().length() != 0)
                    hsqlwhere += " and MatriNo like '%" + matrino.trim() + "%' ";
            } else {
                if(matrino != null && matrino.trim().length() != 0)
                    hsqlwhere = " where MatriNo like '%" + matrino.trim() + "%' ";
            }
            hsql = "from Student " + hsqlwhere;
            Query query = sessionHibernate.createQuery(hsql);
            ArrayList<Student> stuArray = (ArrayList<Student>) query.list();
            HttpServletRequest request = (HttpServletRequest) ActionContext.getContext().get(ServletActionContext.HTTP_REQUEST);
            request.setAttribute("stuArray", stuArray);
        }
        // ---- 交费操作 ----
        if(stuParamArray != null && "update".equals(action)) {
            for(int i = 0; i < stuParamArray.size(); i++) {
                if(stuParamArray.get(i).getPayAmount() != null
                    && stuParamArray.get(i).getPayOK() != null
                    && stuParamArray.get(i).getStudentId() != null) {
                    String hsqlstr = "from Student where StudentId=" +
                        stuParamArray.get(i).getStudentId();
                    Query query = sessionHibernate.createQuery(hsqlstr);
                    ArrayList<Student> stuArray = (ArrayList<Student>) query.list();
                    if(stuArray.size() >= 1) {
                        Student stu = stuArray.get(0);
                        stu.setPayAmount(stuParamArray.get(i).getPayAmount());
                        stu.setPayOK(stuParamArray.get(i).getPayOK());
                        sessionHibernate.save(stu);
                    }
                }
            }
        }
    }
}

```



```

    }
}
}
}
sessionHibernate.getTransaction().commit();
return SUCCESS;
}

/**
 * 此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
 */
}

```

如果是查询数据操作，则根据学生姓名和录取通知书号来生成 HSQL 语句，再做查询，将结果记录列表放入到 request 对象中。

如果是交费操作，哪个学生交多少费用都已经保存到了 Action 的 stuParamArray 属性中，通过如下的语句来判断数据输入是否正确：

```

stuParamArray.get(i).getPayAmount() != null && stuParamArray.get(i).getPayOK() !=
null
&& stuParamArray.get(i).getStudentId() != null

```

也就是说支付金额、是否付清、学生的 ID 号属性值不能为空，否则表明输入的数据有误，不必做数据更新处理。但是这里又不能生成一个学生对象，再设置这个对象值以作持久化，因为对象已经存在了，要做的工作就是设置收费的情况。

```

Student stu = stuArray.get(0);
stu.setPayAmount(stuParamArray.get(i).getPayAmount());
stu.setPayOK(stuParamArray.get(i).getPayOK());
sessionHibernate.save(stu);

```

以上语句，第一句得到了学生对象 stu；第二句得到用户所输入的交易金额；第三句用于设置是否付清学费；第四句即可做持久化处理了。

### 16.3.8 宿舍分配功能的实现

要进行宿舍分配首先就是查询出要分配的学生，这就需要修改 BedAction 类中的代码。

#### BedAction.java

```

package com.csai.action;
import java.util.ArrayList;
import java.util.Map;
import org.hibernate.Query;
import org.hibernate.Session;
import com.csai.POJO.Bedchamber;
import com.csai.POJO.Student;
import com.csai.db.HibernateUtil;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class BedAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String studentname; // 学生姓名
    public String action; // 动作

```



```

public String matrino;//录取通知书号
public ArrayList<Student> stuParamArray;
@Override
public String execute() throws Exception {
    Session sessionHibernate=HibernateUtil.getSession();
    sessionHibernate.beginTransaction();
    String hsqlwhere=new String("");
    String hsql=new String("");
    Map request = (Map)ActionContext.getContext().get("request");
    //----如果是查询操作----
    if("select".equals(action)){
        if(studentname!=null&&studentname.trim().length()!=0)
            hsqlwhere="where studentName like '%" +studentname.trim()+"%' ";
        if(hsqlwhere!=null&&hsqlwhere.length()!=0){
            if(matrino!=null&&matrino.trim().length()!=0)
                hsqlwhere+=" and matriNo like '%" +matrino.trim()+"%' ";
        }else{
            if(matrino!=null&&matrino.trim().length()!=0)
                hsqlwhere=" where matriNo like '%" +matrino.trim()+"%' ";
        }
        hsql="from Student "+hsqlwhere;
        Query query=sessionHibernate.createQuery(hsql);
        ArrayList<Student> stuArray =(ArrayList<Student>)query.list();
        request.put("stuArray", stuArray);
    }
    //----查询出宿舍清单----
    hsql="from Bedchamber";
    Query query=sessionHibernate.createQuery(hsql);
    ArrayList<Bedchamber> bedArray =(ArrayList<Bedchamber>)query.list();
    request.put("bedArray", bedArray);
    //----分配宿舍----
    if(stuParamArray!=null&&"update".equals(action)){
        for(int i=0;i<stuParamArray.size();i++){
            if(stuParamArray.get(i)!=
                null&&stuParamArray.get(i).getPayOK()!=null&&
                stuParamArray.get(i).getPayOK()==1&&
                stuParamArray.get(i).getBedchamber().getBedchamberId()!=null&&
                stuParamArray.get(i).getBedchamber().getBedchamberId()!=0){
                String hsqlstr="from Bedchamber where bedchamberId="
                    +stuParamArray.get(i).getBedchamber().getBedchamberId();
                Query queryBed=sessionHibernate.createQuery(hsqlstr);
                ArrayList<Bedchamber> bedArray1 =
                    (ArrayList<Bedchamber>)queryBed.list();
                if(bedArray1.size()>=1){
                    hsqlstr="from Student where studentId="+
                        stuParamArray.get(i).getStudentId();
                    Query queryStu=sessionHibernate.createQuery(hsqlstr);
                    ArrayList<Student> stuArray =
                        (ArrayList<Student>)queryStu.list();
                    if(stuArray.size()>=1){
                        Student stu=stuArray.get(0);
                        stu.setBedchamber(bedArray1.get(0));
                        sessionHibernate.save(stu);
                    }
                }
            }
        }
    }
}

```



```

    }
    }
    }
    }
    sessionHibernate.getTransaction().commit();
    return SUCCESS;
}
/**
 *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
 */
}

```

如果要做查询操作，先要构造 HSQL 查询语句，再做查询，得到查询结果。虽然查询的结果涉及多个数据库表，但是在 HSQL 语句中并不需要做联合查询，因为还有配置文件做了关系映射，Hibernate 会去维护这些关系。

做宿舍分配时，先得到表示学生的 Student 对象，再得到表示宿舍的 Bedchamber，建立起两者之间的关系，做 Hibernate 持久化操作即可更新学生所在的宿舍了，也为学生分配了宿舍，语句如下所示：

```

Student stu=stuArray.get(0);
stu.setBedchamber (bedArray1.get(0));
sessionHibernate.save(stu);

```

第一句得到 Student 对象；第二句设置学生对象 stu 的宿舍；第三句作持久化处理。

## 16.4 小结

读者通过本章的系统开发动手操作后，应当对 Hibernate 有了更深的认识，并能结合 Struts 和 Hibernate 来开发系统了，也应当能运用 Hibernate 来对已有的 Struts 技术实现的系统做出改进了。

使用 Hibernate 后，程序员的大量工作集中于编写配置文件，Action 类的代码也得到了大大简化，程序逻辑清晰简单，可见 Hibernate 真是开发人员的好助手，用好它将事半功倍，这在大型的系统中将表现得更为突出。

## 16.5 练习

在随书光盘中找到本章的源代码，在 Tomcat 中运行这个 Web 应用，查看源代码，并理解 Struts 2 和 Hibernate 4 集成的思想。

# 17

## Spring 3 框架技术

---

学习 Spring 3 框架技术，最重要的就是掌握它的 IoC 核心技术，再把这种技术运用到实践工程中。虽然运用 Spring 3 就可以直接搭建起 Web 系统的框架，但它如果和 Struts 2 技术结合起来，将具有更好的实用性、易用性。

本章将和读者一起详细剖析 Spring 的 IoC 核心技术，并配以实例讲解，再学会如何整合 Struts 2 和 Spring 3。

### 17.1 Spring 介绍

Spring 是一个开源的框架，它的内容非常丰富，从实现了 MVC 的 Spring Web MVC、表现层的 Spring 标签，到 Spring AOP、核心的 IoC 等，需要理解的概念也非常之多。可以把 Spring 看成是各种开源技术的粘合剂，用它可以把各种组件组装在一起，而又互不影响，因此有可能通过 Spring 满足开发人员对轻量级 J2EE 系统开发的一站式需求。

#### 17.1.1 Spring 的框架结构

Spring 的框架结构如图 17-1 所示，本章重点讲解 Spring 的最为常用的 IoC，以及与 Struts2、Hibernate 4 的集成。

从图 17-1 来看，Spring 的框架结构包含的内容十分广泛，本书也不打算详细介绍，读者可以参见 Spring 3 的专业书籍。下面讲解主要的模块。

(1) 核心容器。核心容器提供 Spring 框架的基本功能。核心容器的主要组件是 **BeanFactory**，它是工厂模式的实现。**BeanFactory** 使用控制反转 (IoC) 模式将应用程序的配置和依赖性规范与实际的应用程序代码分开。

(2) Context。即 Spring 上下文，它实际上是一个配置文件，向 Spring 框架提供上下文信息。Spring 上下文包括企业服务，例如 JNDI、EJB、电子邮件、国际化、校验和调度功能。Spring 3 还提供了新的表达式语言，这是以前的版本中所没有的。



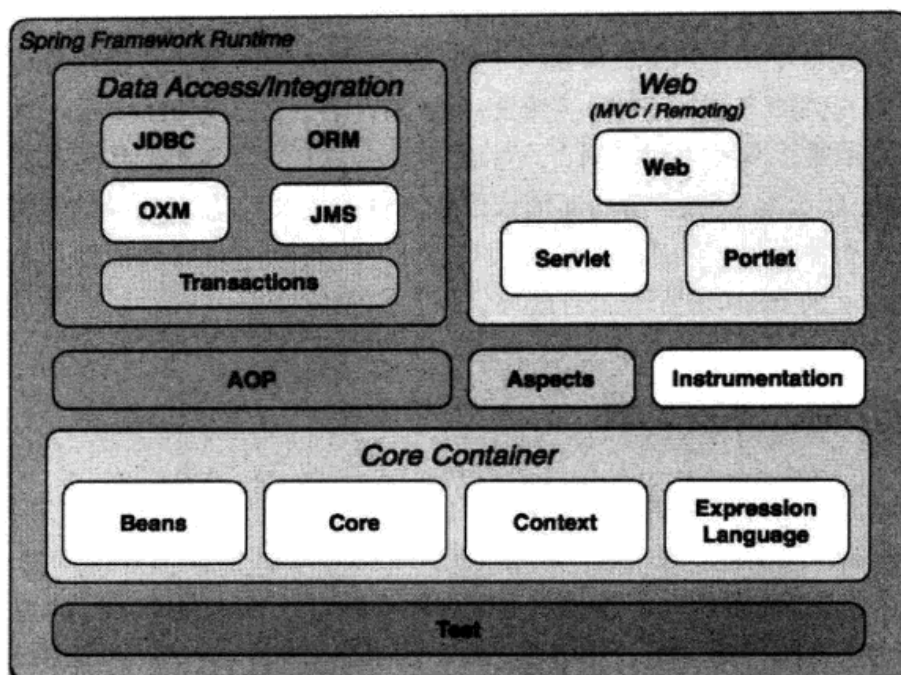


图 17-1 Spring 的框架结构

(3) AOP。Spring AOP 模块直接将面向方面的编程功能集成到了 Spring 框架中。所以可以很容易地使 Spring 框架管理的任何对象支持 AOP。Spring AOP 模块为基于 Spring 的应用程序中的对象提供了事务管理服务。通过使用 Spring AOP，不用依赖 EJB 组件，就可以将声明性事务管理集成到应用程序中。

(4) Aspects。Spring 3 提供了对 AspectJ 框架的整合和应用。

(5) Instrumentation。Instrumentation 提供了类装载的支持，以及用于实现到某些应用的服务器中的类装载器。

(6) Test。该模块提供与测试有关的组件集成，如 JUnit、TestNG。

(7) 数据输入输出与集成。Spring DAO 提供了 JDBC 的抽象层，它可消除冗长的 JDBC 编码和解析数据库厂商特有的错误代码。并且 JDBC 封装包还提供了一种比编程性更好的声明性事务管理方法，不仅仅是实现了特定接口，而且对所有的 POJOs (plain old Java objects) 都适用。Spring 框架插入了若干个 ORM 框架，其中包括 JDO、Hibernate 和 iBatis SQL Map，Spring 和 Hibernate 的结合相当良好。OXM 提供了对象与 XML 映射的抽象，比如 JaxB、Castor 等。

(8) Web。Web 上下文模块建立在应用程序上下文模块之上，为基于 Web 的应用程序提供了上下文，Spring 框架支持与 Struts 的集成。Spring Web MVC 提供了 Web 应用的 MVC 实现，其中容纳了大量的视图技术，其中包括 JSP、Velocity、Tiles、iText 和 POI 等。

### 17.1.2 理解 IoC 与 DI

Spring 的 IoC 组件主要专注于如何利用类、对象和服务去组成一个企业级应用，通过规范的方式，将各种不同的控件整合成一个完整的应用。到底什么是 IoC 呢？这是 Spring 中最为重要的专业术语。

IoC (Inversion of Control, 控制反转) 又称为 DI (Dependency Injection, 依赖注入)。这两个名字表面上看来完全不同, 其实质却是一样的。比如在 A 类的某个方法中新建了一个 B 类的实例, 并需要用到 B 类的方法, 则称 A 类依赖于 B 类, 当 B 类的方法修改时, A 类中的调用代码也需要修改, 如果要想两者之间低耦合, 可以将 A 类中生成 B 类的实例的工作由第三方来完成, 比如 Spring 的 bean 容器。Spring 通过配置文件的配置, 生成 B 类的实例, 然后再注入到 A 类中, 这样 A 类就可以使用了, 因此称为依赖注入, 然而本来这种信赖关系是 A 类来建立的, 现在却由 Spring 的 bean 来管理了, 因此又称为控制反转。

## 17.2 控制反转技术

控制反转技术是 Spring 框架的核心技术, 内涵简单, 外延丰富, 掌握它并不难。

`org.springframework.beans` 及 `org.springframework.context` 包是 Spring IoC 容器的基础。`BeanFactory` 提供的高级配置机制, 使得管理任何性质的对象成为可能。`ApplicationContext` 是 `BeanFactory` 的扩展, 功能得到了进一步增强, 比如更容易与 Spring AOP 集成、消息资源处理(国际化处理)、事件传递及各种不同应用层的 context 实现(如针对 web 应用的 `WebApplicationContext`)。

`ApplicationContext` 完全由 `BeanFactory` 扩展而来, 因而 `BeanFactory` 所具备的能力和行为也适用于 `ApplicationContext`。因此, 在不知道是选择 `BeanFactory` 还是 `ApplicationContext` 时, 不妨使用 `ApplicationContext`。

### 17.2.1 容器的基本原理

`org.springframework.beans.factory.BeanFactory` 是 Spring IoC 容器的核心接口, 它的职责包括: 实例化、定位、配置应用程序中的对象及建立这些对象间的依赖。IoC 容器负责容纳 bean, 并对 bean 进行管理。

Spring 为我们提供了许多易用的 `BeanFactory` 实现, `XmlBeanFactory` 就是最常用的一个。该实现将以 XML 方式描述组成应用的对象以及对象间的依赖关系。Spring IoC 容器的基本原理如图 17-2 所示。

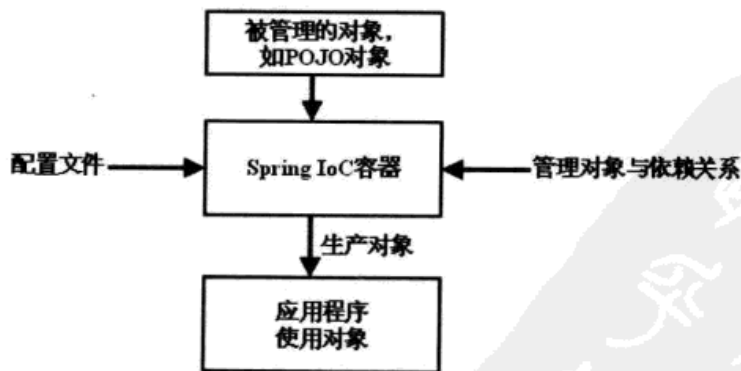


图 17-2 Spring IoC 容器的基本原理



开发人员用配置文件给出 Spring IoC 容器中对象的属性值，以及对象之间的依赖关系；Spring IoC 容器负责管理这些对象和对象之间的依赖关系；当应用程序需要使用对象时，就通过 Spring IoC 容器来得到对象。

### 17.2.2 XML 配置文件格式

Spring 主要支持三种配置格式：XML 格式、Java 属性文件格式或使用 Spring 公共 API 编程实现。由于 XML 元数据配置格式简单明了，且具有良好的可读性，笔者建议读者使用 XML 配置文件，本书中也将使用 XML 配置文件。

以下是一个 XML 配置文件的基本结构：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
  <bean id="..." class="...">
    <!-- 有关 Bean 的属性与依赖配置 -->
  </bean>
  <bean id="..." class="...">
    <!-- 有关 Bean 的属性与依赖配置 -->
  </bean>
  <!-- 更多的 Bean 配置 -->
</beans>
```

<beans>是配置文件的顶底元素，其中可以有一个或多个<bean>，一个<bean>即表示一个 Bean 对象的配置。bean 定义与应用程序中实际使用的对象一一对应。通常情况下 bean 的定义包括：服务层对象、数据访问层对象（DAO）、类似 Struts Action 的表示层对象、Hibernate SessionFactory 对象、JMS Queue 对象等。

### 17.2.3 实例化容器

实例化容器有多种方法，如下所示：

```
Resource resource = new FileSystemResource("beans.xml");
BeanFactory factory = new XmlBeanFactory(resource);
```

或

```
ClassPathResource resource = new ClassPathResource("beans.xml");
BeanFactory factory = new XmlBeanFactory(resource);
```

或

```
ApplicationContext context = new ClassPathXmlApplicationContext(
    new String[] {"applicationContext.xml", "applicationContext-part2.xml"});
BeanFactory factory = (BeanFactory) context;
```

XML 文件需要放置在 CLASSPATH 目录中，也就是 Java 程序能找到的类路径，再以此基准来配置路径。在 Web 应用中，一般 CLASSPATH 会指向“WEB-INF/classes”目录，因此就可以把

XML 配置文件放到这个目录下，如果直接放在“WEB-INF/classes”下，实例化容器时给出 XML 的文件名就可以了，如果放在“WEB-INF/classes”的子目录下，实例化容器时要以“子目录/XML 文件名”的形式给出路径。

### 17.2.4 下载并开发一个简单的 Spring 应用

Spring 的下载地址为：

<http://www.springsource.org/download>

打个这个网址如图 17-3 所示。

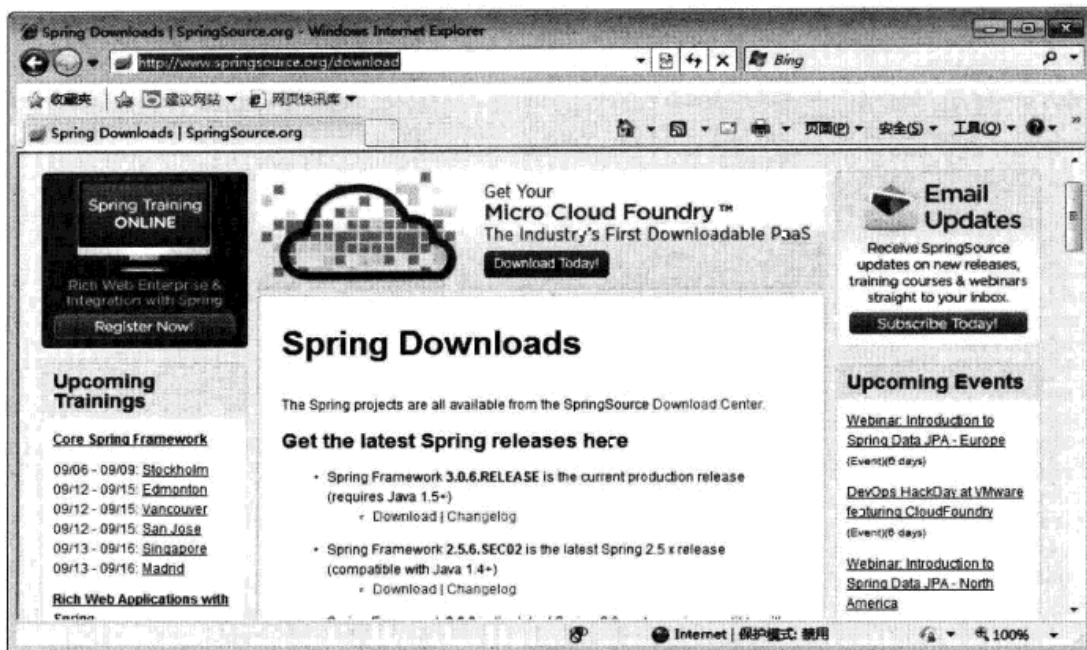


图 17-3 Spring 3 的下载页面

选择相应的压缩包即可下载，本书使用的是 `spring-framework-3.0.5.RELEASE` 版本，请读者自行下载。

Spring 3 与 Spring 2 的组件包的最大区别就是 Spring 3 拆成了许多个 jar 组件包。为简便起见，在新建 Web 应用时，可将 Spring 3 的 `dist` 子目录下的所有以“`org.springframework`”打头的 jar 包全部拷贝过去。

#### 【实例 17-1】一个简单的 IoC 应用示例

本例将新建两个 `Student` 类，并在 XML 配置文件中配置两个 `bean`，接着在 JSP 页面中生成对象并输出对象的属性值。

首先要搭建好开发环境，本章的所有源代码都放在一个名为“`ch17`”的工程中。建好工程后，需要在 Web 应用的“`WEB-INF/lib`”目录下加入 Spring 3 的 `dist` 子目录下的所有以“`org.springframework`”打头的 jar 包，第 16 章的 Web 工程的 `WEB-INF/lib` 目录下的所有 jar 包，再加上 Struts 2 中的 `struts2-spring-plugin-2.2.3.jar` 包。



本例开发完后，应用的情况如图 17-4 所示。

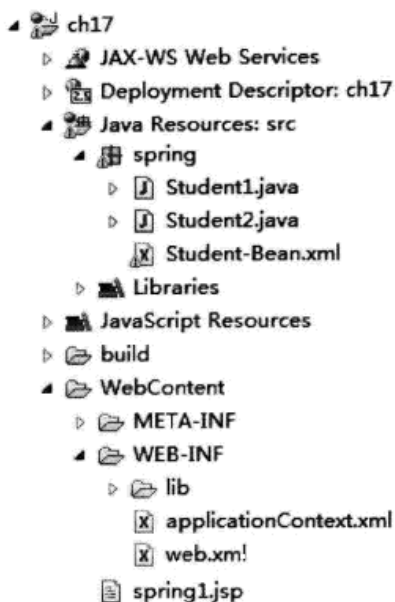


图 17-4 应用的情况

Spring 包中有本例开发的两个 Java 类，以及这两个 Java 类的配置文件，WebContent 中的 spring1.jsp 文件用于调试程序中生成 bean 实例的情况。

#### Student1.java

```
package spring;
public class Student1 {
    public long studentId;
    public String studentName;
    /**
     此处省去以上属性的 getXxxx() 方法和 setXxxx() 方法
    */
}
```

#### Student2.java

```
package spring;
public class Student2 {
    public long studentId;
    public String studentName;
    public Student2(long studentId, String studentName) {
        this.studentId = studentId;
        this.studentName = studentName;
    }
}
```

这两个类有所不同，Student1 类中的代码就是属性与对应的 setXxxx() 方法、getXxxx() 方法；Student2 类中的代码就是属性与一个构造函数。这样方便程序演示也方便 setter 注入和构造函数注入这两种方法。

## Student-Bean.xml

```
<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE beans PUBLIC "-//SPRING/DTD BEAN/EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="student1" class="spring.Student1">
    <property name="studentId">
      <value>1</value>
    </property>
    <property name="studentName">
      <value>邓佳容</value>
    </property>
  </bean>
  <bean id="student2" class="spring.Student2">
    <constructor-arg type="long">
      <value>2</value>
    </constructor-arg>
    <constructor-arg type="java.lang.String">
      <value>黄婧</value>
    </constructor-arg>
  </bean>
</beans>
```

这个配置文件中定义了两个 bean，第一个 bean 对应的类为 Student1，在实例化时将属性 studentId 的值设为“1”，将 studentName 属性值设为“邓佳容”；第二个 bean 对应的类为 Student2，在实例化时利用构造函数将属性 studentId 的值设为“2”，将 studentName 属性值设为“黄婧”。接下来看如何在 JSP 页面中使用。

## spring1.jsp

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="org.springframework.beans.factory.BeanFactory,
org.springframework.beans.factory.xml.XmlBeanFactory,
org.springframework.core.io.ClassPathResource,
org.springframework.core.io.Resource,
spring.Student1, spring.Student2" %>

<html>
<head>
<title>使用 Spring2</title>
</head>
<body>
<%
//直接使用 Student
Student2 student=new Student2(1,"邓佳容");
out.print(student.studentId+": "+student.studentName+"<br>");
//利用 Spring 使用 Student
Resource res=new ClassPathResource("spring/Student-Bean.xml");
BeanFactory factory=new XmlBeanFactory(res);
Student1 student1=(Student1) factory.getBean("student1");
out.print(student1.getStudentId()+" "+student1.getStudentName()+"<br>");
Student2 student2=(Student2) factory.getBean("student2");
out.print(student2.studentId+": "+student2.studentName+"<br>");
%>
</body>
</html>
```



程序中先是使用原始的 `new` 关键字新建了一个 `Student2` 类的实例，再输出 `studentId` 属性和 `studentName` 属性的值；接下来再以 Spring 注入的方式分别分成了 `Student1` 类的实例 `student1`、`Student2` 类的实例 `student2`，接着即输出 `studentId` 属性和 `studentName` 属性的值。注入时，`student1` 采用的是 setter 注入方法，`student2` 采用的是构造函数注入方法。默认情况下，Spring 都会以工厂的形式生成 bean 实例。程序运行结果如图 17-5 所示。

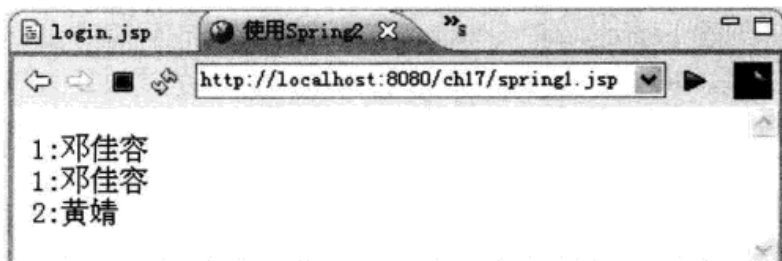


图 17-5 程序运行结果

## 17.2.5 XML 配置文件解析

### 1. 加载多个 XML 配置文件

有时需要将 XML 配置文件分拆成多个部分。为了加载多个 XML 文件生成一个 `ApplicationContext` 实例，可以将文件路径作为字符串数组传给 `ApplicationContext` 构造器。另外一种方法是使用一个或多个 `<import/>` 元素来从另外一个或多个文件加载 bean 定义。所有的 `<import/>` 元素必须放在 `<bean/>` 元素之前以完成 bean 定义的导入。如下是一个示例：

```
<beans>
<import resource="services.xml"/>
  <import resource="resources/messageSource.xml"/>
  <import resource="/resources/themeSource.xml"/>
  <bean id="bean1" class="..." />
  <bean id="bean2" class="..." />
</beans>
```

在上面的例子中，将从三个外部文件：`services.xml`、`messageSource.xml` 及 `themeSource.xml` 来加载 bean 定义。这里采用的都是相对路径，因此，此例中的 `services.xml` 一定要与导入文件放在同一目录或类路径中，而 `messageSource.xml` 和 `themeSource.xml` 的文件位置必须放在导入文件所在目录下的 `resources` 目录中。正如您所看到的那样，开头的斜杠 “/” 实际上可忽略。



**提示** 根据 Spring XML 配置文件的 Schema(或 DTD)，被导入文件必须是完全有效的 XML bean 定义文件，且根节点必须为 `<beans>` 元素。

### 2. 给 bean 命名

每个 bean 都有一个或多个 id，这实际上就是 bean 的名称，这些 id 在 IoC 容器中必须唯一，而 `class` 属性指定实例化对象的类型。可以使用 `id` 或 `name` 属性来指定 bean 的名称，如果没有给 `id` 而给出了 `name`，则会将 `id` 设置为与 `name` 相同的值；此外，还可以通过 `alias` 属性给出别

名。在 XML 配置文件中, 可用单独的<alias/> 元素来完成 bean 别名的定义, 如:

```
<alias name="fromName" alias="toName"/>
```

### 3. 实例化 bean

实例化 bean 的方法主要有三种: 用构造器来实例化, 使用静态工厂方法实例化和使用实例工厂方法实例化。

当采用构造器来创建 bean 实例时, Spring 对 class 并没有特殊的要求, 我们通常使用的 class 都适用。也就是说, 被创建的类并不需要实现任何特定的接口, 或以特定的方式编码, 只要指定 bean 的 class 属性即可。不过根据所采用的 IoC 类型, class 可能需要一个默认的空构造器。

当采用静态工厂方法创建 bean 时, 除了需要指定 class 属性外, 还需要通过 factory-method 属性来指定创建 bean 实例的工厂方法。Spring 将调用此方法 (其可选参数接下来介绍) 返回实例对象。下面的 bean 定义展示了如何通过工厂方法来创建 bean 实例。注意, 此定义并未指定返回对象的类型, 仅指定该类包含的工厂方法。在此例中, createInstance() 必须是一个 static 方法。

```
<bean id="exampleBean" class="examples.ExampleBean2"
    factory-method="createInstance"/>
```

与使用静态工厂方法实例化类似, 用来进行实例化的实例工厂方法位于另外一个已有的 bean 中, 容器将调用该 bean 的工厂方法来创建一个新的 bean 实例。为使用此机制, class 属性必须为空, 而 factory-bean 属性必须指定为当前 (或其祖先) 容器中包含工厂方法的 bean 的名称, 而该工厂 bean 的工厂方法本身必须通过 factory-method 属性来设定, 来参看以下的例子。

```
<!-- 包含有被调用的 createInstance() 方法的工厂 bean -->
<bean id="myFactoryBean" class="...">
    ...
</bean>
<!-- 通过工厂 bean 创建的 bean -->
<bean id="exampleBean" factory-bean="myFactoryBean"
    factory-method="createInstance"/>
```

bean 的属性及构造器参数既可以引用容器中的其他 bean, 也可以是内联 (inline, 在 spring 的 XML 配置中使用<property>和<constructor-arg>元素定义) bean。<value>元素通过字符串来指定属性或构造器参数的值。

在<constructor-arg>或<property>元素内部还可以使用 ref 元素。该元素用来将 bean 中指定属性的值设置为对容器中的另外一个 bean 的引用。如前所述, 该引用 bean 将被作为依赖注入, 而且在注入之前会被初始化。通过 id/name 指向另外一个对象主要有两种形式。

第一种形式也是最常见的形式, 是通过使用<ref>标记指定 bean 属性的目标 bean, 通过该标签可以引用同一容器或父容器内的任何 bean (无论是否在同一 XML 文件中)。XML “bean” 元素的值既可以是指定 bean 的 id 值也可以是其 name 值。

```
<ref bean="someBean"/>
```

第二种形式是使用 ref 的 local 属性指定目标 bean, 要求所引用的 bean 存在同一文件中。local 属性值必须是目标 bean 的 id 属性值。如果在同一配置文件中没有找到引用的 bean, XML



解析器将抛出一个异常。

```
<ref local="someBean"/>
```

#### 4. 集合

通过<list>、<set>、<map>及<props>元素可以定义和设置与 Java Collection 类型对应 List、Set、Map 及 Properties 的值。

```
<bean id="moreComplexObject" class="example.ComplexObject">
  <!-- java.util.Properties, 用户的邮件列表, key 为用户名, value 为邮箱地址 -->
  <property name="adminEmails">
    <props>
      <prop key="administrator">dzycsai@csai.cn</prop>
      <prop key="support">dhl@csai.cn</prop>
    </props>
  </property>
  <!-- java.util.List -->
  <property name="someList">
    <list>
      <value>a list element followed by a reference</value>
      <ref bean="myDataSource"/>
    </list>
  </property>
  <!-- java.util.Map -->
  <property name="someMap">
    <map>
      <entry>
        <key>
          <value>key1 设置</value>
        </key>
        <value>value1 设置</value>
      </entry>
      <entry>
        <key>
          <value>key2 设置</value>
        </key>
        <ref bean="myDataSource"/>
      </entry>
    </map>
  </property>
  <!-- java.util.Set -->
  <property name="someSet">
    <set>
      <value>值</value>
      <ref bean="myDataSource"/>
    </set>
  </property>
</bean>
```

### 17.2.6 使用容器

BeanFactory 提供的方法极其简单, 它仅提供了 6 种方法供调用:

(1) `boolean containsBean(String)`: 如果 `BeanFactory` 包含给定名称的 `bean` 定义 (或 `bean` 实例), 则返回 `true`。

(2) `Object getBean(String)`: 返回以给定名字注册的 `bean` 实例。根据 `bean` 的配置情况, 如果为 `singleton` 模式将返回一个共享的实例, 否则将返回一个新建的实例。如果没有找到指定的 `bean`, 该方法可能会抛出 `BeansException` 异常 (实际上将抛出 `NoSuchBeanDefinitionException` 异常), 在对 `bean` 进行实例化和预处理时也可能抛出异常。

(3) `Object getBean(String, Class)`: 返回以给定名称注册的 `bean` 实例, 并转换为给定 `class` 类型的实例, 如果转换失败, 相应的异常 (`BeanNotOfRequiredTypeException`) 将被抛出。上面的 `getBean(String)` 方法也适用该规则。

(4) `Class getType(String name)`: 返回给定名称的 `bean` 的 `class`。如果没有找到指定的 `bean` 实例, 则抛出 `NoSuchBeanDefinitionException` 异常。

(5) `boolean isSingleton(String)`: 判断给定名称的 `bean` 定义 (或 `bean` 实例) 是否为 `singleton` 模式, 如果 `bean` 没找到, 则抛出 `NoSuchBeanDefinitionException` 异常。

(6) `String[] getAliases(String)`: 返回给定 `bean` 名称的所有别名。

`singleton` 指的是单模式, 这是默认情况下的模式, 在这种情况下, 容器生成 `bean` 时, 对于相同 `id` 的 `bean`, 使用共享的空间, 也就是说如果用 “`==`” 判断生成的两个 `id` 相同的 `bean`, 会得到 `true`。

## 17.3 集成 Struts 2、Hibernate 4 与 Spring 3

集成 Struts 2、Hibernate 4 和 Spring 3 是相当简单的, 这两个开源的框架作为搭档构建系统将演绎优美的组合。但是需要开发人员来考虑整合的目的是什么?

Spring 最核心的技术就是 `IoC` 了, 在整合时当然要充分利用起来了。整合 Struts 和 Spring 有两种策略:

(1) Spring 管理 Struts 的控制器。也就是说 `Action` 交由 Spring 来管理, 利用 `IoC` 的特性为 `Action` 注入业务逻辑组件。

(2) Spring 管理业务逻辑组件, 需要时即注入。

也可以结合以上两种策略。总的来说, Spring 就像一个产生组件对象的工厂, 可以源源不断地输出组件对象, 注入到需要之处, 而开发人员需要做的就是编写组件类的代码和编写配置文件。

### 17.3.1 集成前的环境准备

集成 Struts 2 和 Spring 3 前需要准备好开发环境, 主要工作内容也就是如下两部分。

(1) 导入所需的组件包。用 Struts 2 搭建好开发环境后, 还需要将 Spring 的组件包放到 Web 应用的 “`WEB-INF/lib`” 目录中, 包括 Spring 3 的 `dist` 子目录下的所有以 “`org.springframework`” 打头的 `jar` 包。





**提示** 其实 jar 组件包放到 CLASSPATH 指出的路径中就可以了，但是为了保证 Web 应用的可移植性，一般将 jar 组件包放到 Web 应用的“WEB-INF/lib”目录中。

(2) 修改相应的配置文件。主要是修改 web.xml 文件，此文件位于 Web 应用的 WEB-INF 目录下，在此文件中加入如下的配置：

```
<!-- 采用 Listener 完成 Spring 容器的初始化-->
<listener>
<listener-class>
    org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
```

或

```
<!-- 利用参数来加载配置文件-->
<context-param>
    <param-name>contextCofigLoction</param-name>
    <param-value>WEB-INF/bean1.xml,WEB-INF/bean2.xml</param-value>
</context-param>
```

第一种形式采用 Listener 完成 Spring 容器的初始化，为什么要初始化呢？在实例 17-1 中的 JSP 页面代码中读者可能注意到了，生成一个 bean 组件，首先就要加载 XML 文件，再根据 XML 文件的配置结合 Spring 来生成。能否在 Web 应用中自动生成 bean 组件呢？可以在 Web 应用启动服务时就加载配置，这样就不必再在应用程序中烦琐地加载配置文件了。

在第一种形式的情况下，系统默认会到 Web 应用的 WEB-INF 目录下去查找 applicationContext.xml 配置文件，并加载，因此开发人员可以把 bean 组件的配置写在这个配置文件中。

第二种形式可以用来加载多个配置文件，文件之间用“,”分隔并初始化 Spring 容器，但注意参数名一定要是 contextCofigLoction。



**提示** 其实要加载多个配置文件，在 applicationContext.xml 配置文件使用 <import> 标签也能实现。

如果以上两种形式的配置都有，会先找第一种形式对应的 applicationContext.xml 配置文件，如果找不到再找第二种形式下的其他配置文件，如果还是找不到则 Spring 无法正常初始化。



**提示** 如果 Web 服务器不支持 Servlet 2.3 以上的规范，使用以上两种形式的配置将不能初始化 Spring 容器，此时可以采用 load-on-startup Servlet 来实现。

### 17.3.2 集成示例与剖析

#### 【实例 17-2】用 Struts 2+Hibernate 4+Spring 3 集成改进用户登录功能

Spring 的长处在于它的 IoC 机制，可以大大降低系统的耦合程度，提高系统的可移植性。

通过在 Spring 的配置文件中做出设置，Spring 容器就像是一个对象工厂，能源源不断地为程序生成对象，也能管理这些对象，这样程序中就不必再出现新建对象（new 语句）这样的语句了，而可以直接使用对象。那么如何来生成这些对象呢？运用 Spring 就可以通过构造函数、属性或静态工厂的方法来声明对象，而由 Spring 注入对象。

在用户登录实例中业务逻辑的代码集中在 Action 类的 LoginAction 中，在 LoginAction 类的 execute() 方法中代码相对较多，在分层式系统的设计中，应当尽量封装程序逻辑，比如对 SysUser 类的所有操作应当封装成一个单独的类（即封装业务逻辑），这样在 Action 类中只要简单调用即可。

本例将把 Struts 2、Hibernate 4、Spring 3 这三种框架技术整合起来，各司其职，充分运用各自的特长，协同工作。

Struts 与 Hibernate 的整合方法与策略在实例 15-6 中读者已经体会过了，那么如何将 Spring 加入进来呢？有如下的整合思路：

（1）将 Struts 2 的 Action 对象交由 Spring 来生成、管理，Struts 2 不再管理 Action 对象，这样可以利用 Spring 3 的依赖注入机制动态生成 Action 对象，从而将系统的各种类、程序逻辑通过配置文件中的设置交织在一起。

（2）由 Spring 3 来生成、管理 Hibernate 4 的会话，从而方便对数据库中的数据进行操作。根据以上两个整合思路，结合使用，如图 17-6 所示。

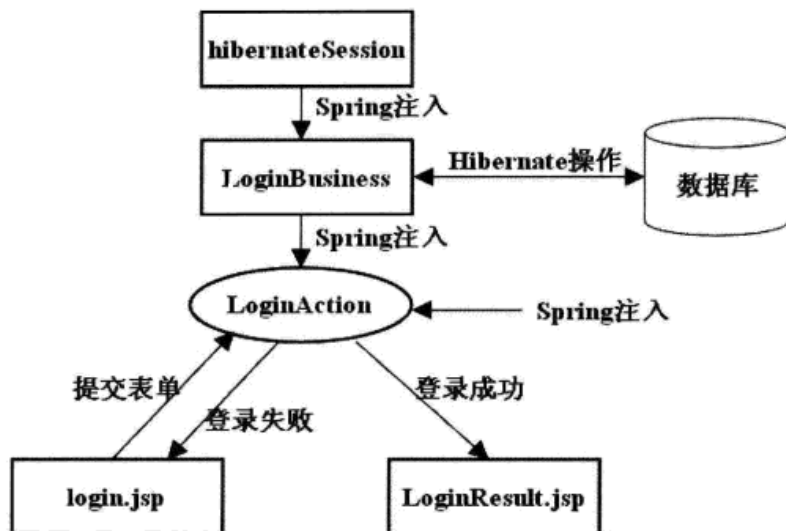


图 17-6 用户登录功能中 Struts 2、Hibernate 4 与 Spring 3 的整合思路

在用户登录界面，提交表单后，将由 Spring 来生成 LoginAction 对象；LoginAction 对象要使用到封装了登录验证业务逻辑的 LoginBusiness 对象，则 LoginBusiness 对象由 Spring 注入；LoginBusiness 对象要通过得到 hibernateSession 对象来操作数据，即通过 Hibernate 与数据库会话对象，这个 hibernateSession 对象也由 Spring 注入。

Struts、Hibernate 与 Spring 的集成要使用到 Spring 相关的组件包，请将 struts2-spring-plugin-2.2.3.jar 组件包也放到当前工程的“WebContent/WEB-INF/lib”目录下（使用 Windows 操作系统进行复制操作即可），所需的 jar 组件包与实例 17-1 中的相同，可参考实



例 17-1 中所述的详细操作。

struts2-spring-plugin-2.2.3.jar 组件包位于 Struts 2 的 lib 目录中,如果要集成 Struts 2 与 Spring 就必须用到这个组件包,需要注意的是,随着 Struts 版本的不断更新,文件的版本可能稍有不同,不过没有关系,依样操作即可。



**提示** 请确保已在应用中放入了 Struts 2 和 Hibernate 4 的组件包。



**提示** 将 jar 组件包放入到当前 Web 应用中后,请刷新工程,以确保 Eclipse 能找到并使用组件包。刷新的方法是在工程名称上单击右键,在快捷菜单中选择“Refresh”,或选中工程名称后按 F5 键。

#### LoginBusiness.java

```
package com.csai.business;
import java.util.ArrayList;
import org.hibernate.Query;
import org.hibernate.Session;
public class LoginBusiness {
    public Session sessionHibernate;
    //验证用户名和密码是否正确
    public boolean loginCert(String username,String password){
        //开始事务
        sessionHibernate.beginTransaction();
        //创建 HQL 查询
        Query hqlQuery=sessionHibernate.createQuery("from SysUser
            where username=? and password=?");
        //设置查询语句中的参数值
        hqlQuery.setString(0,username);
        hqlQuery.setString(1,password);
        //得到查询结果
        ArrayList result =(ArrayList)hqlQuery.list();
        //结束事务
        sessionHibernate.getTransaction().commit();
        //判断验证是否通过
        if(result.size()>0)
            return true;
        else
            return false;
    }
    public Session getSessionHibernate() {
        return sessionHibernate;
    }
    public void setSessionHibernate(Session sessionHibernate) {
        this.sessionHibernate = sessionHibernate;
    }
}
```

LoginBusiness 类有一个方法 loginCert(), 用于验证用户名和密码是否正确, 如果正确则返回 true, 错误则返回 false。代码中使用到了 sessionHibernate 属性, 却没有为 sessionHibernate



赋值就直接使用了，这就是利用了 Spring 的特性——依赖注入，读者可结合下一步的 `applicationContext.xml` 配置文件中的内容来看就清楚了。

读者也可以注意到，`LoginBusiness` 类有一个属性叫 `sessionHibernate`，且带有 `getSessionHibernate()` 方法和 `setSessionHibernate()` 方法，这些都是 Spring 通过属性来注入对象的前提。

#### LoginAction.java

```
package com.csai.action;
import com.csai.business.LoginBusiness;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String username;//用户名
    public String password;//密码
    public String message;//返回 JSP 页面的消息
    public LoginBusiness log;//业务逻辑封装类
    @Override
    public String execute() throws Exception {
        if(log.loginCert(username,password)){
            message=new String("用户名和密码输入正确,通过验证。");
            return SUCCESS;
        }else{
            message=new String("用户和密码输入有误!");
            return INPUT;
        }
    }
    /**
     *此处省略以上属性对应的 setXxxx() 方法和 getXxxx() 方法
     */
}
```

这个类的代码已经相当简单了，就像是一个 POJO 类，里面的代码大多是属性的声明以及属性相应的 `set`、`get` 方法，在 `execute()` 方法中直接调用了 `log` 属性的 `loginCert()` 方法来做登录验证，为什么可以直接调用而不需要使用 `new` 语句来新建对象呢？这也是因为使用了 Spring 的注入机制。在下一步还会详细讲解配置文件中的配置。

#### web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns=
"http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/
javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
    <display-name>Struts info</display-name>
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
        </filter-class>
    </filter>
    <filter-mapping>
```



```

        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <filter>
        <filter-name>struts-cleanup</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.ActionContextCleanUp
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts-cleanup</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
</web-app>

```

在配置文件中增加了一个<listener>配置项，表示一个监听器，对应的监听器类为org.springframework.web.context.ContextLoaderListener，这样的话一旦 Web 应用部署就会自动加载 Spring 的配置文件，默认情况下为 Web 应用的 WEB-INF 目录下的 applicationContext.xml 文件。

struts.xml 配置文件内容如下。

#### struts.xml

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.objectFactory" value="spring" />
    <package name="ch17" namespace="" extends="struts-default">
        <action name="Login" class="login">
            <result>/loginResult.jsp</result>
            <result name="input">/login.jsp</result>
        </action>
    </package>
</struts>

```

在配置文件中增加了一个常量配置，如下所示：

```
<constant name="struts.objectFactory" value="spring" />
```

有了此项配置后，Struts 的对象工厂将由 Spring 容器来承担。此外还将名为 Login 的 Action 的配置项 class 属性值设为 login，login 是 Spring 配置文件中一个 bean 的名称，这样一旦用户提交请求时，就会循着 Spring 的配置文件去找到 bean 的定义，从而生成对象。

applicationContext.xml 配置文件位于“WebContent/WEB-INF”目录下，与 web.xml 配置文



件处于同一个目录中。

applicationContext.xml

```
<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE beans PUBLIC "-//SPRING/DTD BEAN/EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean id="sessionHibernate" class="com.csai.db.HibernateUtil"
        factory-method="getSession"/>
    <bean id="loginBusiness" class="com.csai.business.LoginBusiness">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
    <bean id="login" class="com.csai.action.LoginAction" singleton="false">
        <property name="log" ref="loginBusiness"/>
    </bean>
</beans>
```

login 这个 bean 对应着 com.csai.action.LoginAction, singleton 属性为 false 表示不使用单例模式, 其中有个属性名为 log (读者可参见 LoginAction 的代码), 参考了 loginBusiness 这个 bean。

loginBusiness 的配置在上文中有, 对应着 com.csai.business.LoginBusiness, 其也有个属性, 名为 sessionHibernate, 参考了 sessionHibernate 这个 bean。这样 login 中的 log 属性和 loginBusiness 中的 sessionHibernate 属性都采用了属性注入的方式。

sessionHibernate 对象由 com.csai.db.HibernateUtil 生成, 其中生成对象的方法是 getSession, 这从 sessionHibernate 这个 bean 中的配置可以看出。

至此, Struts 2、Hibernate 4 与 Spring 3 集成完毕, 您可以运行程序了。

可能有读者觉得代码量似乎更大了, 是的, 因为我们现在练习的是一个小的实例, 但如果您的系统相对更大一些, 就会感受到整合的优势了, 比如代码的重用、架构的稳定性、系统的可维护性等。

默认情况下 Spring 使用的是 applicationContext.xml 配置文件, 如果要使用其他的配置文件可在 web.xml 文件中参考如下的配置方法:

```
<!-- 利用参数来加载配置文件-->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>WEB-INF/bean1.xml,WEB-INF/bean2.xml</param-value>
</context-param>
```

<param-value>标签中指出了配置文件的清单, 配置文件之间用逗号(,)相隔, 路径是相对 Web 应用的路径。

注入对象的方法本例中使用了两种: 一种是通过属性注入, 详见 LoginAction 中的 log 属性及其配置; 另一种是通过静态工厂方法注入, 详见 LoginBusiness 类中的 sessionHibernate 属性及其配置。还有一种就是通过构造函数来注入, 使用形式如下:

```
<bean id="名称" class="对应的类">
    <constructor-arg type="数据类型">
        <value>值</value>
    </constructor-arg>
    <constructor-arg type="数据类型">
```



```
<ref bean="另一个 bean 的名称"/>
</constructor-arg>
<constructor-arg type="数据类型">
  <ref local="另一个 bean 的名称"/>
</constructor-arg>
</bean>
```

以上形式中表示使用了带三个参数的构造函数来注入对象，第一个参数为直接设定其值，第二个参数可以引用同一容器或父容器内的任何 bean（无论是否在同一 XML 文件中）；第三个参数使用 ref 的 local 属性指定目标 bean，要求所引用的 bean 是存在同一文件中。

## 17.4 小结

Spring 的 IoC 组件主要专注于如何利用类、对象和服务去组成一个企业级应用，通过规范的方式，将各种不同的控件整合成一个完整的应用。控制反转又称为依赖注入。

org.springframework.beans.factory.BeanFactory 是 Spring IoC 容器的核心接口，它的职责是实例化、定位、配置应用程序中的对象及建立这些对象间的依赖。整合 Spring 与 Struts 的过程中可以充分利用 Spring 的 IoC 核心技术。

## 17.5 练习

自行设计一个数据库表，表示某个信息系统的用户，参照实例 17-2 实现用户登录功能。



# 18

## 基于 SSH 实现 报到管理系统

---



本章中将采用 Struts 2+Hibernate 4+Spring 3 技术来继续改进第 16 章中实现的报到管理系统。读者从这个项目案例中汲取到一些系统架构设计的思想，以及 Struts、Spring、Hibernate 这些流行的开源框架的集成策略。一起来看如何进行改进。

### 18.1 系统设计思想

到现在为止，我们已经一起学习了 Struts、Hibernate、Spring 这三种流行的框架技术了，Struts 的优点在于实现了 MVC 模式，将 Web 系统各组件进行了良好的分工合作；Spring 的特性在于 IoC 机制；Hibernate 的长处在于数据持久化。要结合这三个优点，结合的策略有许多种，比如可以将 Hibernate 的 sessionFactory、数据操作组件交给 Spring 容器来管理，必要时进行注入处理；可以将 Struts 的 Action 交给 Spring 容器来管理，而不必再在 Action 中声明业务逻辑操作的组件了。

#### 18.1.1 改进思路

现将采用如下的思路来改进用 Struts 2+Hibernate 4 实现的报到管理系统：

- ① 将所有的 Action 交由 Spring 3 容器来管理。
- ② 利用 Spring 3 的 IoC 特性，将 Hibernate 4 的数据库操作会话交由 Spring 3 来注入。

依据以上两点思路，改进以后的系统架构如图 18-1 所示。

这样改进是基于以下的原因：

- (1) Action 类中需要用到数据操作类，比如 AdminUserAction 中要用到 AdminUser，此时可以将 AdminUser 的实例作为 LoginAction 的属性，再在 applicationContext 中做出配置，以做注



入处理。当然，是否采用这种方式注入，读者在实践工程中可根据需要自行决定。

(2) 数据操作类中都要用到 `sessionHibernate`，每次持久化数据时都要重复地得到会话，可以将会话对象作为数据操作类的属性，再在 `applicationContext` 中做出配置，以做注入处理。

经过以上改进后，会发现代码数量并没有减少多少，但是系统模块与模块之间的耦合性降低了，互相之间比较独立，层次结构清晰明了。

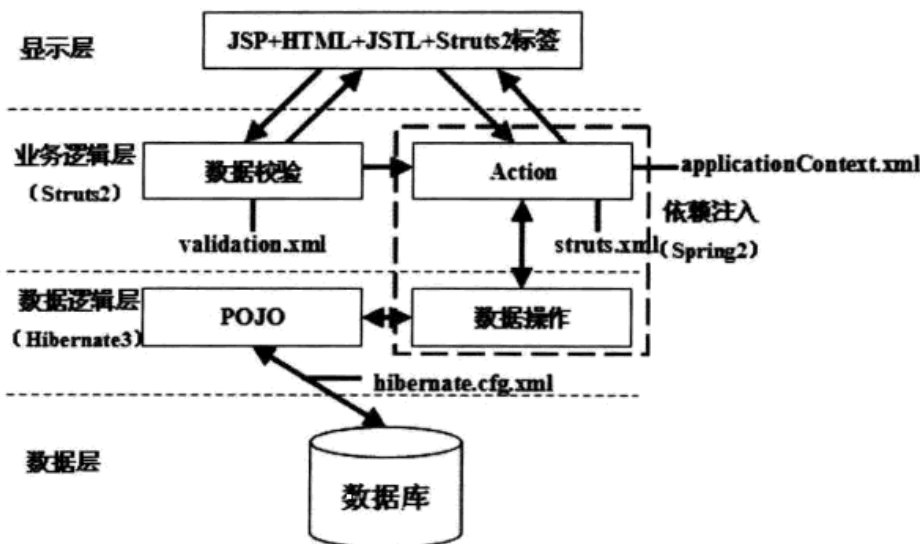


图 18-1 基于 SSH 改进系统后的架构

### 18.1.2 系统配置文件

系统的配置文件主要有三个：`web.xml`、`hibernate.cfg.xml`、`applicationContext.xml`。其中 `web.xml`、`hibernate.cfg.xml` 原来已有，只是需要修订；`applicationContext.xml` 要新增到 Web 应用的 `WEB-INF` 目录下。

`web.xml`

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  ... .. (此处省去一些原来已有的配置)
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
</web-app>
  
```

此 `web.xml` 与第 16 章的 `web.xml` 相比，不同之处就在于增加了监听器的配置，这样 Web 应用在启动时就会自动加载 Spring 的默认配置文件 `applicationContext.xml` 中的组件配置。

## struts.xml

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.objectFactory" value="spring" />
    <package name="ch18" namespace="" extends="struts-default">
        <!-- ===== 用户登录功能 ===== -->
        <action name="Login" class="login">
            <result>/index.jsp</result>
            <result name="input">/login.jsp</result>
        </action>
        <!-- ===== 基础数据管理模块 ===== -->
        <action name="RegStatus" class="regStatus">
            <result>/basicdata/regstatus.jsp</result>
        </action>
        <action name="AdminUser" class="adminUser">
            <result>/basicdata/adminuser.jsp</result>
        </action>
        <action name="Bedchamber" class="bedchamber">
            <result>/basicdata/bedchamber.jsp</result>
        </action>
        <action name="Class" class="classAction">
            <result>/basicdata/class.jsp</result>
        </action>
        <action name="Speciality" class="speciality">
            <result>/basicdata/specialityadmin.jsp</result>
        </action>
        <action name="Matri" class="matri">
            <result>/basicdata/matri.jsp</result>
            <result name="input">/login.jsp</result>
        </action>
        <!-- ===== 报到分班管理模块 ===== -->
        <action name="ClassAdmin" class="classAdmin">
            <result>/class/classadmin.jsp</result>
            <result name="input">/class/classadmin.jsp</result>
        </action>
        <action name="ClassView" class="classView">
            <result>/class/classview.jsp</result>
        </action>
        <!-- ===== 收费管理模块 ===== -->
        <action name="AcceptMoney" class="acceptMoney">
            <result>/money/acceptmoney.jsp</result>
        </action>
        <!-- ===== 宿舍分配管理模块 ===== -->
        <action name="Bed" class="bed">
            <result>/bed/bedchamber.jsp</result>
        </action>
    </package>
</struts>

```

在以上配置中，每个<action>标签的 name 属性值都没有改变，这样表示层的代码就可以不



做变更；只是修改了 class 属性值，这个值对应着 applicationContext.xml 文件中的<bean>标签的 id 属性值。

applicationContext.xml

```
<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE beans PUBLIC "-//SPRING/DTD BEAN/EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean id="sessionHibernate" class="com.csai.db.HibernateUtil"
        factory-method="getSession"/>
    <bean id="login" class="com.csai.action.LoginAction" singleton="false">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
    <bean id="regStatus" class="com.csai.action.RegStatusAction" singleton="false">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
    <bean id="adminUser" class="com.csai.action.AdminUserAction" singleton="false">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
    <bean id="bedchamber" class="com.csai.action.BedchamberAction" singleton="false">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
    <bean id="classAction" class="com.csai.action.ClassAction" singleton="false">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
    <bean id="speciality" class="com.csai.action.SpecialityAction" singleton="false">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
    <bean id="matri" class="com.csai.action.MatriAction" singleton="false">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
    <bean id="classAdmin" class="com.csai.action.ClassAdminAction" singleton="false">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
    <bean id="classView" class="com.csai.action.ClassViewAction" singleton="false">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
    <bean id="acceptMoney" class="com.csai.action.AcceptMoneyAction" singleton="false">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
    <bean id="bed" class="com.csai.action.BedAction" singleton="false">
        <property name="sessionHibernate" ref="sessionHibernate"/>
    </bean>
</beans>
```

从这个配置文件来看，主要就是在每个数据操作类中注入 sessionHiberante 属性，这样做的好处是在 Action 中不必再行创建数据操作类的实例；在数据操作类中也不必再创建 Session 的实例，直接使用即可，Spring 会自动注入，使得系统具有良好的解耦性。



## 18.2 系统实现

不失一般性，以 AcceptMoneyAction 为例，源代码如下：

AcceptMoneyAction.java

```
package com.csai.action;
import java.util.ArrayList;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.ServletActionContext;
import org.hibernate.Query;
import org.hibernate.Session;
import com.csai.POJO.Student;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class AcceptMoneyAction extends ActionSupport {
    private static final long serialVersionUID = 1L;
    public String studentname;
    public String action;
    public String matrino;
    public ArrayList<Student> stuParamArray;
    public Session sessionHibernate;
    @Override
    public String execute() throws Exception {
        sessionHibernate.beginTransaction();
        //----查询数据操作----
        String hsqlwhere=new String("");
        String hsql=new String("");
        if("select".equals(action)){//如果是查询操作
            if(studentname!=null&&studentname.trim().length()!=0)
                hsqlwhere="where StudentName like '%" +studentname.trim()+"%' ";
            if(hsqlwhere!=null&&hsqlwhere.length()!=0){
                if(matrino!=null&&matrino.trim().length()!=0)
                    hsqlwhere+=" and MatriNo like '%" +matrino.trim()+"%' ";
            }else{
                if(matrino!=null&&matrino.trim().length()!=0)
                    hsqlwhere=" where MatriNo like '%" +matrino.trim()+"%' ";
            }
            hsql="from Student "+hsqlwhere;
            Query query=sessionHibernate.createQuery(hsql);
            ArrayList<Student> stuArray =(ArrayList<Student>)query.list();
            HttpServletRequest request =(HttpServletRequest)ActionContext
                .getContext().get(ServletActionContext.HTTP_REQUEST);
            request.setAttribute("stuArray", stuArray);
        }
        //----交费操作----
        if(stuParamArray!=null&&"update".equals(action)){
            for(int i=0;i<stuParamArray.size();i++){
                if(stuParamArray.get(i).getPayAmount()!=
```



```

        null&&stuParamArray.get(i).getPayOK()!=
        null&&stuParamArray.get(i).getStudentId()!=null){
            String hsqlstr="from Student where StudentId="
                +stuParam Array.get(i).getStudentId();
            Query query=sessionHibernate.createQuery(hsqlstr);
            ArrayList<Student> stuArray =(ArrayList<Student>)query.list();
            if(stuArray.size()>=1){
                Student stu=stuArray.get(0);
                stu.setPayAmount(stuParamArray.get(i).getPayAmount());
                stu.setPayOK(stuParamArray.get(i).getPayOK());
                sessionHibernate.save(stu);
            }
        }
    }
}
sessionHibernate.getTransaction().commit();
return SUCCESS;
}
/**
 此处省去了属性的 getXxxx()方法和 setXxxx()方法
 */
}

```

从代码中可以发现，execute()方法中原有的以下语句：

```
Session sessionHibernate=HibernateUtil.getSession();
```

已经不再需要了，而在 Action 中多出了一个属性 sessionHibernate，该属性由 Spring 3 窗口来注入，execute()方法中只需直接使用 sessionHibernate 属性即可。

其他 Action 的改进就不再赘述了，原理都是一样的，具体代码读者可参见随书光盘中的内容。

## 18.3 小结

读者通过本章的系统开动手操作后，应当对 Hibernate 有了更深的认识，并能结合 Struts 和 Hibernate 来开发系统了，也能运用 Hibernate 对已有的 Struts 技术实现的系统做出改进了。

使用 Hibernate 后，程序员的大量工作集中于配置文件的编写，Action 类的代码也得到了大大简化，程序逻辑清晰简单，可见 Hibernate 真是开发人员的好助手，用好它将事半功倍，这在大型的系统中将表现得更为突出。

基于 Struts 2、Hibernate 4 和 Spring 3 结合实现的通用在线文章系统仍然将系统分成四层：显示层、业务逻辑层、数据逻辑层、数据层，其中 Spring 负责注入组件及组件管理工作。

## 18.4 练习

在随书光盘中找到本章的系统源代码，在 Eclipse 中调试程序，查看源代码，进一步熟悉系统的改进思想。